Technology
Arts Sciences
TH Köln

Technische Hochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften

M A S T E R A R B E I T

Konzeption und Entwicklung eines auf DeepSpeech basierenden Open Source Editors zur Transkription von Audio- und Videomaterial

Vorgelegt an der TH Köln
Campus Gummersbach
im Studiengang
Informatik Computer Science (Ma.) Software Engineering

ausgearbeitet von:
MAURICE BARTOSZEWICZ
(Matrikelnummer:)

Erstprüfer: Prof. Dr. Heide Faeskorn-Woyke
Zweitprüfer: Prof. Dr. Dietlind Zühlke

Gummersbach, im Februar 2021

Zusammenfassung

Immer häufiger müssen und sollten gesprochene Informationen in schriftlicher Form verfügbar sein. Dafür versuchen verschiedene Transkriptionsprogramme den Nutzer beim Abtippen des Ausgangsmaterials mit verschiedenen Annehmlichkeiten zu unterstützen. Eine Vielzahl von Online-Diensten geht sogar noch einen Schritt weiter und liefert gegen eine Gebühr eine fertige, automatisch erstellte Transkription. Da die Gebühren für den Privatanwender sehr teuer sein können und die Online-Dienste aus datenschutzrelevanten Gründen nicht immer verwenden werden dürfen, ist das Ziel dieser Arbeit die Implementierung einer offenen offline Alternative.
Diese Alternative soll ein Open Source Editor auf Basis der freien Speech-To-Text-Engine DeepSpeech sein und dem Nutzer einerseits eine offline Transkription zur Verfügung stellen und andererseits ihn bei der Korrektur dieser unterstützen. Um dieses Ziel zu erreichen, wird zuerst die herkömmliche Spracherkennung und schließlich DeepSpeech beschrieben. Darauf aufbauend erfolgt danach die Konzeption und Implementierung des Editors. Da es sich hierbei ausdrücklich um ein Open-Source-Projekt handelt, wird im letzten Teil die Veröffentlichung genauer beleuchtet.

Abstract

More and more often, spoken information must and should be available in written form. For this purpose, various transcription programs try to support the user with various conveniences when transcribing the source material. A variety of online services go one step further and provide a ready-to-use, automatically generated transcription for a fee. Since the fees can be very expensive for the individual user and the online services may not always be used for privacy reasons, the goal of this work is to implement an open offline alternative.
This alternative should be an open source editor based on the open speech-to-text-engine DeepSpeech and should on one hand provide the user with an offline transcription and on the other hand support him in correcting it. To achieve this goal, first the traditional speech recognition and eventually DeepSpeech will be described. This is followed by the conception and implementation of the editor. Since this project is explicitly intended to be an open source project, the last part will take a closer look at the release.

Inhaltsverzeichnis

Abbildungsverzeichnis **6**

Tabellenverzeichnis **9**

Programmausdrücke **10**

1 Einleitung **11**

2 Automatische Spracherkennung **13**

2.1 Herausforderungen 14

2.1.1 Stetige Entwicklung der Sprache 14

2.1.2 Mehrdeutigkeit 14

2.1.3 Aussprache 14

2.1.4 Umgebung 15

2.1.5 Wortschatz 15

2.1.6 Zusammenfassung 16

2.2 Funktionsweise 16

2.3 Bewertung von Spracherkennungssystemen 19

3 DeepSpeech **21**

3.1 Gründe für die Entwicklung 21

3.1.1 Offene Alternative 21

3.1.2 Offline Support 22

3.1.3 Machine Learning Ansatz 23

3.2 Gründe für die Verwendung 23

3.3 Wie funktioniert DeepSpeech? 25

3.4 Beispiel Verwendung 28

4 Konzeption des Editors **29**

4.1 Brainstorming 29

4.2 Test anderer Transkriptionsprogramme 30
4.2.1 easytranscript 30
4.2.2 oTranscribe 31
4.2.3 f4transkript 32
4.2.4 autoEdit 33
4.2.5 Weitere Transkriptionsmöglichkeiten 34
4.3 Erkenntnisse aus dem Test 35
4.3.1 Wertung 35
4.3.2 Häufige Funktionen 35
4.3.3 Besondere Funktionen 36
4.4 Anforderungen 36
4.5 Wireframes 39
4.6 Architektur 41
4.6.1 Das Plug-in-Architekturmuster 41
4.6.2 Übersicht 44

5 Programmiersprache und Plattformwahl **45**
5.1 Wahl des GUI-Frameworks 45
5.2 Media-Player Beispiel 46

6 Ergebnisse **48**
6.1 Ein Projekt öffnen oder erstellen 48
6.2 Eine Transkription bearbeiten 49
6.3 Textbausteine 51
6.4 Wort-für-Wort-Bearbeitung 51
6.5 Einstellungen und Lizenzen 52

7 Schwächen und Fehler in der Transkription **53**

8 Plug-ins **55**
8.1 Plug-in-Struktur 55

8.1.1 Plug-in Interface 56
8.1.2 Plug-in-Manager 58
8.1.3 Beispiel Plug-ins 61
8.2 Plug-ins zur Verbesserung des Ergebnisses 63
8.2.1 Wort 63
8.2.2 Wort-zu-Zahl-Konvertierung 63
8.2.3 Satzzeichen 64
8.2.4 Automatische Großschreibung 64
8.2.5 DeepSpeech Alternativen 65
8.2.6 Vosk Alternativen 65
8.2.7 Umlaute 65
8.2.8 LanguageTool 66
8.3 Plug-ins zur Auswertung und Weiterverarbeitung der Transkription 66
8.3.1 Zusammenfassung 66
8.3.2 Anonymisierung 67
8.3.3 Lesbarkeitsindex 67

9 Problem: Forced Alignment **68**

10 Open Source **69**
10.1 Lizenz 69
10.2 Readme 70
10.3 Contributing 72
10.4 Projektname 76
10.5 Code 76

11 Zusammenfassung und Fazit **77**

12 Ausblick **80**

Referenzen **81**

A Anhang **89**

A.1 Brainstorming 89

A.2 Komponentendiagramm 90

A.3 Editor-Maske 91

A.4 Fehlerarten des Beispieltextes 92

A.5 Einführung / Anleitung 93

A.6 Readme 104

A.7 Contributing 106

Erklärung über die selbständige Abfassung der Arbeit **108**

Abbildungsverzeichnis

1 Einsatzmöglichkeiten des SICARE pilot. (Zur besseren Darstellung gedreht) Quelle: [98, S.105] . . . 13

2 Drei Äußerungen des Wortes „Donau“ von einem Sprecher. Quelle: [29, S.14] . . . 15

3 Drei Äußerungen des Wortes „Donau“ von drei verschiedenen Sprechern. Quelle: [29, S.14] . . . 15

4 Die Architektur eines auf HMM basierenden Spracherkennungssystems. Quelle: [38, S. 201] . . . 18

5 Ausschnitt der bereits unterstützten Sprachen auf Common-Voice. Quelle: [68], (Stand: 23.09.2020) . . . 22

6 Traditionelle Spracherkennungsysteme im Vergleich zur Vorstellung von DeepSpeech. Angepasste Darstellung von: [46, 47-48min] . . . 23

7 DeepSpeech Architektur. Quelle: [66] . . . 25

8 Beispiel Ergebnis des neuralen Netzes. Je intensiver der Grauton, desto höher die Wahrscheinlichkeit. Quelle: [45] . . . 26

9 Beispiel Sequenzen aus der Matrix. Quelle: [45] . . . 27

10 Beispiel Anwendung des CTC-Algorithmus. Quelle: [45] . . . 27

11 Mögliche Ergebnis-Transkriptionen. Quelle: [45] . . . 27

12 Brainstorming: Ideen und Anforderungen . . . 29

13 easytranscript: Hauptmaske . . . 31

14 oTranscribe: Hauptmaske . . . 32

15 f4transkript: Hauptmaske . . . 33

16 autoEdit: Hauptmaske . . . 34

17 Wireframe: Eine neue Transkription erstellen oder öffnen . . . 39

18 Wireframe: Editor . . . 40

19 Wireframe: Zusätzliche Fenster . . . 40

20 Wireframe: Korrektur . . . 41

21 Sequenzdiagramm eines typischen Systems nach der Plug-in-Architekur. Quelle: [41, S. 317] . . . 42
22 Klassendiagramm einer Plug-in-Architektur mit einem einzigen Interface. Quelle: [41, S. 316] . . . 43
23 Komponentendiagramm des zu entwickelnden Editors . . . 44
24 Einfacher Media-Player mittels PySide2 . . . 46
25 Start-Maske . . . 48
26 Ein-Projekt-erstellen-Maske . . . 48
27 Editor-Maske . . . 49
28 Menüleiste: Beispiel der klassischen Operationen . . . 50
29 Menüleiste und Toolbar . . . 50
30 Editor-Maske: Mediensteuerung . . . 50
31 Textbaustein-Maske . . . 51
32 Wort-für-Wort-Bearbeitung . . . 51
33 Einstellungs-Maske . . . 52
34 Lizenz-Maske . . . 52
35 Beispiel: Rechtschreibkorrektur in der Wort-für-Wort-Bearbeitung . . . 55
36 Beispiel: Toolbar-Plug-ins . . . 55
37 Sequenzdiagramm zur Visualisierung der get_word_action-Methode . . . 57
38 Sequenzdiagramm zur Visualisierung der get_toolbar_action-Methode . . . 58
39 Plug-in-Verzeichnis-Struktur . . . 59
40 Wort-Plug-in . . . 63
41 Wort-Plug-in Dialog . . . 63
42 Wort-zu-Zahl-Plug-in . . . 63
43 Sonderzeichen-Plug-in . . . 64
44 DeepSpeech alternativen Plug-in . . . 65
45 Vosk alternativen Plug-in . . . 65
46 LanguageTool Plug-in . . . 66
47 Zusammenfassung Plug-in . . . 66
48 Anonymisierung (Fake) Plug-in . . . 67

49 Lesbarkeitsindex Plug-in 67
50 Readme: Einführung 70
51 Readme: Installation und Verwendung 71
52 Readme: Vorinstallierte Plug-ins 71
53 Readme: Abschluss 72
54 Contributing: Einleitung 73
55 Contributing: Grundregeln 74
56 Contributing: Erster Beitrag 74
57 Contributing: Ein Pull Request erstellen 75
58 Contributing: Ein eigenes Plug-in schreiben 75
59 Contributing: Fehler und Features vorschlagen oder melden 76
60 Brainstorming: Ideen und Anforderungen 89
61 Komponentendiagramm des zu entwickelnden Editors 90
62 Editor-Maske 91

Tabellenverzeichnis

1 Preise verschiedener Speech-To-Text-Services (Stand: 27.11.2020) 11

2 Zusammenfassung der Herausforderungen und dem daraus resultierenden Schwierigkeitsgrad. Vergleiche Tabelle 3.2 von [29, S. 18] 16

3 Beispiel Wörterbuch. Quelle: [48, S. 6] 18

4 Beispiel der Fehlerarten. Quelle: [48, S. 2] 19

5 Verpflichtende Anforderungen 37

6 Nicht verpflichtende Anforderungen 38

7 Farbcodierung / Legende für den von DeepSpeech transkribierten Text 54

8 Weitere Methoden des Plug-in-Managers 60

9 Fehlerarten der Transkription des gesprochenen Schmiergeld-Artikels. [109], [57] 92

Programmausdrücke

1 Installation von DeepSpeech mittels pip 28
2 Verwendung von DeepSpeech unter Python, angelehnt an [43], [75] 28
3 Media-Player umgesetzt mittels PySide2 (gekürzt) 47
4 Plug-in-Interface 56
5 Import einer Python-Datei als Modul 59
6 word_to_number-Plug-in (gekürzt) 61
7 umlauts-Plug-in 62

1 Einleitung

Gesprochene Informationen müssen und sollten immer häufiger in schriftlicher Form verfügbar sein. Darunter vor allem Untertitel, Protokolle aus Meetings und Versammlungen, geschäftliche Audioaufnahmen wie Schriftsätze in Anwaltskanzleien oder auch Befunde im medizinischen Bereich. Deshalb versuchen Transkriptionsprogramme wie zum Beispiel „oTranscribe“ [82], „easytranscript“ [106] oder auch „f4transkript“ [5] den Nutzer während der Transkription durch Tastenkombinationen, Anpassung der Geschwindigkeit, Textbausteine und andere Annehmlichkeiten zu unterstützen. Der Nutzer erhält also Hilfe während des Abtippens. Andere Dienste wie „AmberScript“ [4], „Dragon“ [78], „f4x“ [6] oder auch „DictaNet“ [18] versuchen hingegen (zum Teil zusätzlich) Spracherkennung einzusetzen, um dem Nutzer den Großteil der Arbeit durch eine automatische Transkription zu ersparen.

Problematisch hierbei sind jedoch die zum Teil erheblichen Kosten. Wie Tabelle 1 zu entnehmen ist, sind große Anbieter wie Google Cloud Speech-To-Text [11] oder IBM Watson Speech to Text [50] am kostengünstigsten. Die Anbieter f4x und AmberScript unterscheiden sich nicht nur durch ihren deutlich höheren Preis, sondern auch anhand der Abrechnung. Während Google Cloud Speech-To-Text und IBM Watson Speech to Text die fast genaue Dateilänge zur Abrechnung verwenden, muss hier Volumen in Stunden gekauft werden. Bei AmberScript mindestens eine, bei f4x mindestens fünf Stunden.

Service	**Kosten p. Minute**	**Kosten p. Monat**	**Einmalige Kosten**	**Hinweis und Quelle**
Google Cloud Speech-To-Text	0 €	—	—	Bis 60 Minuten [12]
Google Cloud Speech-To-Text	$0,013$ €	—	—	Bei mehr als eine Millionen Minuten [12]
IBM Watson Speech to Text	$0,017$ €	—	—	[51]
Google Cloud Speech-To-Text	$0,020$ €	—	—	Ab 60 bis eine Millionen Minuten [12]
f4x	$0,2$ €	—	—	Kleinste Einheit: 5h [7]
AmberScript	$0,\overline{3}$ €	—	—	Kleinste Einheit: 1h [4]
DictaNet	—	29 €	—	Mindestlaufzeit 36 Monate [19]
Dragon Home Edition	—	—	159 €	[78]

Tabelle 1: Preise verschiedener Speech-To-Text-Services (Stand: 27.11.2020)

Mit 159,00 € ist damit die Dragon Home Edition besonders im Vergleich zu AmberScript und f4x eine auf Dauer gesehene kostengünstigere Lösung für den Privatanwender. Im Gegensatz zu den bereits vorgestellten Diensten, handelt es sich bei Dragon nicht um einen Online-Dienst. Die Online-Dienste haben erfahrungsgemäß das Problem, dass sie aus datenschutzrelevanten Gründen nicht in jedem Kontext eingesetzt werden können. Durch seine Vertragslaufzeit und den hohen monatlichen Kosten richtet sich DictaNet besonders an den professionellen und kommerziellen Anwender.

Deshalb ist das Ziel dieser Masterarbeit die Konzeption und Entwicklung eines Open Source Editors, bei welchem der Großteil der Transkription durch DeepSpeech übernommen und der Nutzer durch verschiedene Plug-ins bei der anschließenden Korrektur unterstützt wird. DeepSpeech ist dabei eine frei verfügbare Speech-To-Text-Engine, welche eine offline Transkription mittels verschiedener Machine Learning Techniken ermöglicht. Durch die verschiedenen Plug-ins soll ein modularer Ansatz geschaffen werden, mit welchem die Transkription von DeepSpeech automatisch aufbereitet, verbessert oder anderweitig weiterverarbeitet und analysiert werden kann. Die eigentliche Arbeit des Abtippens soll also für den Nutzer durch kostenlose und Privatsphäre-schützende Automation verringert werden.

Hierfür erfolgt zunächst eine Einführung in automatische Spracherkennungssysteme. Dafür werden Herausforderungen, Funktionsweisen und Bewertung dieser erläutert. Im nächsten Kapitel erfolgt darauf aufbauend der Vergleich zu DeepSpeech. Dazu werden die Gründe für die Entwicklung von DeepSpeech wie auch die Gründe für die Verwendung in diesem Projekt beleuchtet. Schließlich werden die Funktionsweise und der praktische Einsatz von DeepSpeech beschrieben.

Darauf folgt die eigentliche Konzeption des Editors. Hierzu werden zuerst das durchgeführte Brainstorming sowie verschiedene andere Transkriptionsprogramme beschrieben und untersucht. Aus diesen gesammelten Erkenntnissen folgen danach die Anforderungen, die ersten Wireframes und die Architekturbeschreibung des Systems. In den nächsten Kapiteln erfolgt sodann die Umsetzung. Dafür wird zuerst die Wahl des GUI-Frameworks erläutert und anschließend die Ergebnisse der Implementierung des Editors vorgestellt. Um etwaige Anforderungen an die Plug-ins und an die Plug-in-Struktur aufzudecken, wird im nächsten Kapitel eine Beispieltranskription von DeepSpeech untersucht und analysiert. Der letzte Teil der Umsetzung beschäftigt sich mit den Plug-ins. Zuerst wird hier die Plug-in-Struktur und danach die aus den gefundenen Fehlern und definierten Anforderungen resultierenden Plug-ins vorgestellt.

Da das Projekt Open Source getrieben ist, wird für die Veröffentlichung im darauffolgenden Kapitel erläutert, welche Dokumente für ein Repository angelegt werden sollten und welche Informationen sie in diesem Projekt beinhalten. Zuletzt erfolgt das Fazit über das Projekt sowie ein Ausblick.

2 Automatische Spracherkennung

Die automatische Spracherkennung (engl. Automatic Speech Recognition, kurz ASR) ist ein Teilgebiet der Computerlinguistik und beschäftigt sich mit der Rekonstruktion von Wörtern aus dem Gesprochenen [29, S. 15]. Neben der Transkription gibt es noch viele weitere Möglichkeiten, automatische Spracherkennung einzusetzen. Darunter zum Beispiel

- Informationsverteilung und Terminabsprache über das Telefon,
- Steuerung verschiedenster Geräte und Maschinen, besonders in Situationen in denen
  - die Hände bereits andere Aufgaben erfüllen (Chirurg, PKW),
  - die Geräte aufgrund anderer Einflüsse nicht erreichbar oder sichtbar sind,
  - die Person körperlich eingeschränkt ist [98, S. 73, 85, 105], [48, S. 331], [29, S. 17].

Ein Beispiel dafür ist der SICARE pilot. Hierbei handelt es sich um eine Fernbedienung, mit welcher verschiedene Geräte mithilfe der Sprache gesteuert werden können. Besonders für Personen mit körperlichen Einschränkungen ist dies eine sinnvolle Alternative zur Verwendung einer Tastatur mit einem Mundstab [98, S. 104]. Abbildung 1 zeigt hierbei verschiedene Einsatzmöglichkeiten.

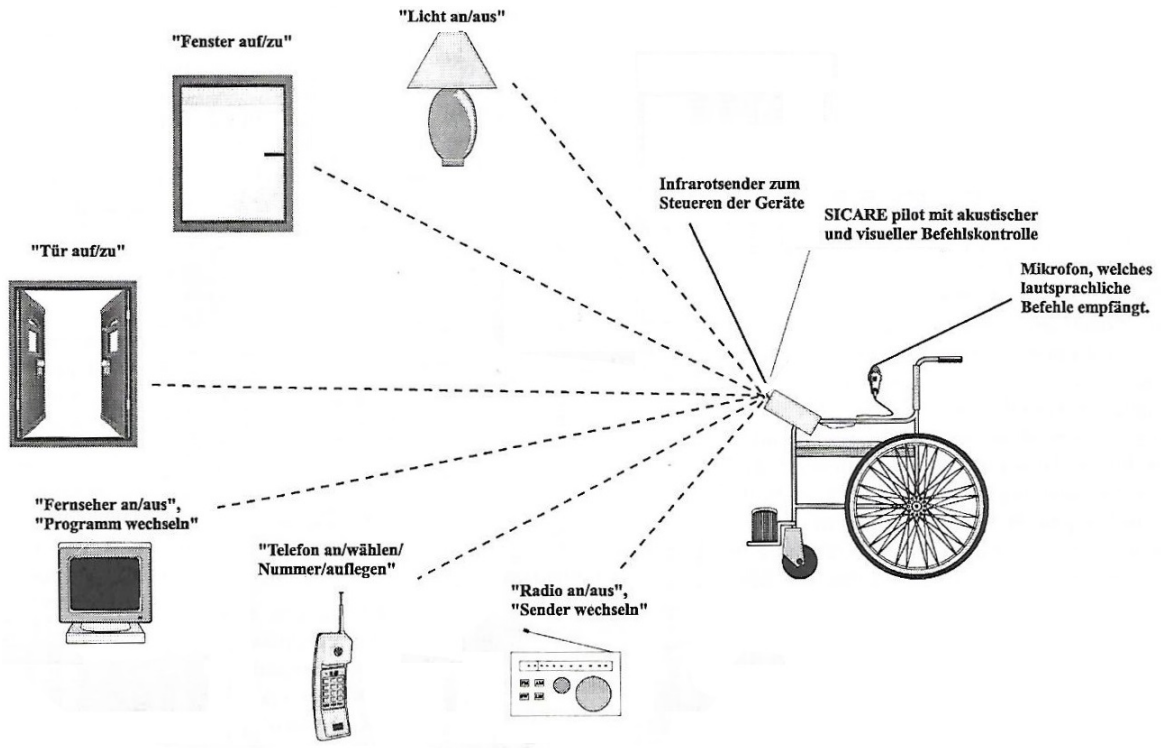


Abbildung 1: Einsatzmöglichkeiten des SICARE pilot. (Zur besseren Darstellung gedreht) Quelle: [98, S.105]

2.1 Herausforderungen

Gesprochene Sprache automatisch zu erfassen birgt einige Probleme und Herausforderungen. Folgend sollen diese genauer erläutert werden.

2.1.1 Stetige Entwicklung der Sprache

Sprachen unterliegen einem stetigen Wandel. Das bedeutet unter anderem, dass Wörter neu entstehen oder gänzlich aus dem Sprachgebrauch verschwinden. Auch werden immer häufiger Wörter aus anderen Sprachen aufgenommen sowie umfunktioniert oder zweckentfremdet. Sprachen sind also lebendig. Daraus resultiert, dass diese nicht durch eine Handvoll Regeln beschrieben werden können, da immer wieder neue Regeln dazu kommen oder verändert werden. [29, S.2], [98, S.28], [23]

2.1.2 Mehrdeutigkeit

Eine weitere Herausforderung ist die Mehrdeutigkeit eines Wortes. Einerseits können verschiedene Wörter mehrere Bedeutungen haben. Ein einfaches Beispiel wäre hierfür „verzogen“. Es kann sich dabei zum einen um das Umziehen an einen anderen Wohnsitz und zum anderen um eine schlechte Erziehung handeln. Andererseits können aber verschiedene Wörter die gleiche oder eine sehr ähnliche Aussprache besitzen. Diese Wörter werden auch „Homophone“ genannt. Ein einfaches Beispiel wären hier die Worte „lehren“ und „leeren“. Es ist also wichtig, den jeweiligen Kontext zu betrachten. [29, S. 2-3 u. 10], [110], [10]

2.1.3 Aussprache

Für die automatische Spracherkennung stellt die Aussprach der einzelnen Wörter und Sätze eine große Herausforderung dar. Neben sehr offensichtlichen unterschiedlichen Aussprachen, bedingt durch Akzente und Dialekte, werden Wörter auch von verschiedenen Personen im gleichen Dialekt unterschiedlich ausgesprochen (siehe Abbildung 3). Es unterscheiden sich beispielsweise Betonung, Lautstärke, Intensität und Dauer. Die unterschiedliche Aussprache hängt dabei von Emotionen, Gesundheitszustand, Müdigkeit sowie den unterschiedlichen physiologischen Eigenschaften ab. Selbst der Zweck der Aufnahme beeinflusst das Sprachverhalten. So ist eine an ein System gerichtete Sprache in der Regel weniger komplex als beispielsweise die Sprache in einer Konversation. Es wird deshalb häufig versucht, die Spracherkennungssysteme an eine einzige Person anzupassen, auch wenn selbst diese, wie in Abbildung 2 deutlich wird, ein Wort niemals hundertprozentig gleich ausspricht. Ist ein System an einen einzigen Sprecher

angepasst, so spricht man von einem sprecherabhängigen, ansonsten von einem sprecherunabhängigen System. [48, S. 1-2], [29, S. 3, 9, 11–14, 18–19], [101, S. 25, 328–329]

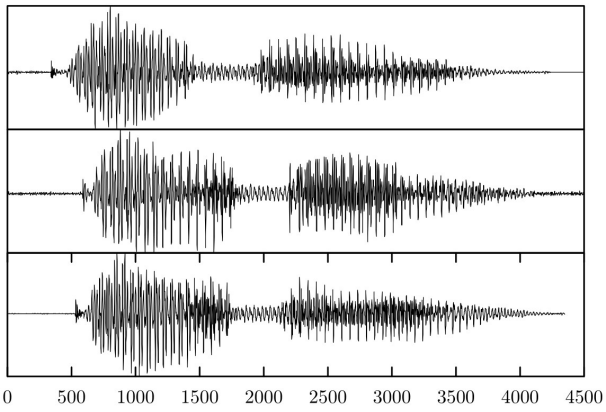


Abbildung 2: Drei Äußerungen des Wortes „Donau“ von einem Sprecher.
Quelle: [29, S.14]

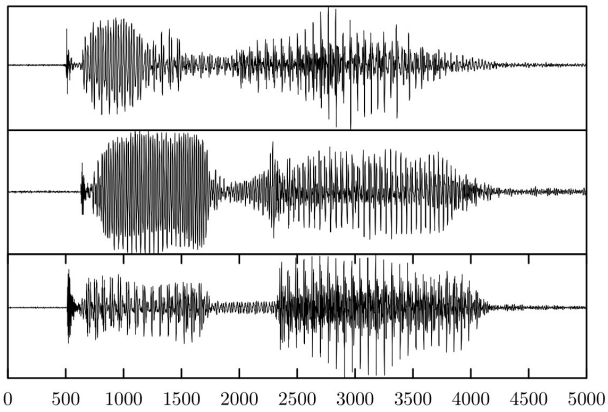


Abbildung 3: Drei Äußerungen des Wortes „Donau“ von drei verschiedenen Sprechern.
Quelle: [29, S.14]

2.1.4 Umgebung

Die Umgebung der Aufnahme spielt auch eine wichtige Rolle. So sind besonders Umgebungsgeräusche, wie beispielsweise andere Konversationen nicht förderlich für die Auswertung. Um etwaige Störgeräusche zu minimieren, wird deshalb ein Headset-Mikrofon (mit einer gewissen Qualität) empfohlen, da hier das Mikrofon näher am Mund ist. [48, S. 2], [29, S. 18]

2.1.5 Wortschatz

Je kleiner die Anzahl der zu erkennenden Wörter ist, desto einfacher sind die Spracherkennungsprobleme zu lösen. Dies liegt einerseits daran, dass seltener Erkennungsfehler entstehen und andererseits die Lösung in vielen Aspekten, wie beispielsweise der benötigten Rechenleistung, deutlich kompakter werden. Dementsprechend ist es leichter einzelne Worte mit einem kleinen Wortschatz zu erkennen, als mehrere Sätze mit einem großen. Ein kleiner Wortschatz entspricht dabei rund 100 Wörtern, ein großer 100.000. [48, S. 1], [29, S.18], [101, S.28, 333]

2.1.6 Zusammenfassung

Folgend eine Zusammenfassung der Herausforderungen und dem daraus resultierenden Schwierigkeitsgrad.

	Einfach	**Schwierig**
Mehrdeutigkeit	Wenige mehrdeutige Wörter und Homophone	Viele mehrdeutige Wörter und Homophone
Aussprache	Gleichartig	Sehr unterschiedlich
Umgebung	Keine Umgebungsgeräusche	Viele Umgebungsgeräusche
Wortschatz	Kleiner Wortschatz	Großer Wortschatz

Tabelle 2: Zusammenfassung der Herausforderungen und dem daraus resultierenden Schwierigkeitsgrad. Vergleiche Tabelle 3.2 von [29, S. 18]

2.2 Funktionsweise

In der Regel unterscheidet man bei Spracherkennungssystemen zwischen der Spracherkennung mit Mustervergleich und der statistischen Spracherkennung. Bei der Spracherkennung mittels Mustervergleich wird das gesprochene Wort mit einer Sammlung von Referenzmustern der zu erkennenden Wörter verglichen. Dafür wird beim Vergleich eine Distanz berechnet, welche den Unterschied angibt. Je unterschiedlicher das gesprochene Wort zu dem Wort des Referenzmusters ist, desto größer ist diese Distanz. Das Wort mit der kleinsten Distanz ist schließlich das erkannte Wort. [101, S. 28, 353]

Diese Art der Spracherkennung eignet sich am besten für die Erkennung einzeln gesprochener Worte eines Sprechers, da sonst viele verschiedene Referenzmuster nötig sind. Zur genaueren Betrachtung der Variabilität hat sich deshalb ein statistischer Ansatz, besonders für die kontinuierliche Spracherkennung, als erfolgreicher herausgestellt. [101, S. 28, 353, 365]

Für die statistische Spracherkennung müssen verschiedene Schritte durchlaufen werden. Euler definiert beispielsweise in [29, S. 20] die folgenden Schritte:

1. Geräuschreduktion, Quellentrennung
2. Merkmalsextraktion
3. Akustische Modelle
4. Wortlexikon
5. Statistisches Sprachmodell
6. Syntax
7. Semantik
8. Pragmatik

Bei der Geräuschreduktion bzw. der Quellentrennung gilt es, etwaige Hintergrundgeräusche aus dem Aufnahmesignal zu entfernen und nur den Teil, in dem gesprochen wird, zu extrahieren. Da sich dies häufig als recht kompliziert herausstellt, wird hier eher versucht, nur sehr deutliche Störgeräusche herauszufiltern und etwaige Grenzen zum Gesprochenen großzügig zu wählen. Ebenfalls werden in diesem Schritt Frequenzen herausgefiltert, welche für das menschliche Ohr nicht hörbar sind. [29, S. 29-30], [46]

Für die Merkmalsextraktion wird das Sprachsignal in 5 bis 25ms lange Abschnitte (sogenannte Fenster) aufgeteilt. Danach werden aus den jeweiligen Fenstern verschiedene numerische Merkmale extrahiert und in einem sogenannten Merkmalsvektor festgehalten. Dieser Vektor enthält beispielsweise das Frequenzspektrum sowie die Energie des jeweiligen Fensters. [48, S. 3-4]

In der Sprache ist ein Laut (Phon) die kleinstmögliche Einheit. Ein Phon kann durch einen Dialekt, eine Emotion oder durch andere äußeren Umstände unterschiedlich klingen. Solange jedoch dieser Lautunterschied keinen Bedeutungsunterschied hervorruft, können die unterschiedlichen Phone zu einem Phonem zusammengefasst werden. Phone, die zu einem gemeinsamen Phonem gehören, können in einem Wort beliebig ausgetauscht werden, ohne die Bedeutung des Wortes zu verändern. Ein einfaches Gegenbeispiel dafür sind die Phone [r] und [l]. So verändert sich die Bedeutung des Wortes „Ratte“ zu „Latte“ beim Austausch. [29, S. 9-10]

Mithilfe eines akustischen Modells, welches durch ein Hidden Markov Modell (HMM) realisiert wird, kann schließlich herausgefunden werden, welche Phone am wahrscheinlichsten durch die Merkmalsvektoren dargestellt werden. Vereinfacht gesagt werden in einem HMM Beobachtungen mittels Wahrscheinlichkeiten mit internen Zuständen verknüpft. Im Falle der Spracherkennung sind die Merkmalsvektoren die Beobachtung, die Variabilität der Aussprache die Wahrscheinlichkeiten und die internen Zustände die Phone. Es wird deshalb auch „Laut-HMM“ [101, S. 405] genannt. [48, S. 4-6], [101, S. 405]

Ein Teil des akustischen Modells ist das sogenannte Wörterlexikon oder Wörterbuch. In diesem werden die Wörter und ihre Phonem-Kombinationen festgehalten. Anhand der erkannten Phone kann darauf aufbauend das eigentliche Wort identifiziert werden. Es spezifiziert also, welche Wörter im Spracherkennungssystem erkannt werden können. Der Vorteil eines Laut-HMM in Kombination mit einem Wörterbuch ist, dass dieses sehr leicht um neue Wörter erweitert werden kann. Es muss lediglich das Wort in phonetischer Umschrift bekannt sein. Ein einfaches Beispiel lässt sich in Tabelle 3 finden. [48, S. 4-6], [101, S. 405]

Wort	Phonem-Kombination
BLINDS	B L AY N D Z
CLOSE	K L OW S
CLOSE	K L OW Z

Tabelle 3: Beispiel Wörterbuch. Quelle: [48, S. 6]

Wie bereits in Kapitel 2.1.2 erläutert, sind Homophone ein Problem in der Spracherkennung. Um diesem Problem entgegenzuwirken, wird ein statistisches Sprachmodell eingesetzt. Dieses Sprachmodell gibt die Wahrscheinlichkeit einer Wortfolge in der jeweiligen Sprache an. Wenn also nach dem akustischen Modell, mehrere Wörter infrage kommen, kann das statische Modell helfen, das wahrscheinlichste Wort anhand seiner benachbarten Wörter zu finden. [48, S. 6], [101, S. 429]

Dafür werden in der Regel N-Gramm-Modelle verwendet. Ein N-Gramm besteht dabei aus mehreren N-großen Wortfolgen. Je größer N ist, desto größer ist der benötigte Speicherplatz und die mögliche Erkennungsrate. In der Praxis werden mittlerweile auch 4- bis 5-Gramm-Modelle eingesetzt. Falls die Wortfolge im höchsten N-Gramm-Modell nicht gefunden werden kann, erfolgt eine Suche im nächst kleineren. Der große Vorteil von N-Gramm-Modellen ist, dass diese einfach mit Texten der jeweiligen Sprache trainiert werden können und somit kein Wissen über die Sprache benötigt wird. [48, S. 6], [101, S. 428]

Aus den bisherigen Schritten resultiert der extrahierte Text. Die Architektur des beschriebenen Spracherkennungssystems ist in Abbildung 4 visualisiert. Das Ergebnis kann nun optional hinsichtlich der Grammatik (Syntax), dem Sinn und Inhalt (Semantik) und den nicht-wörtlichen Äußerungen (Pragmatik) analysiert werden. [80], [79], [101, S. 9]

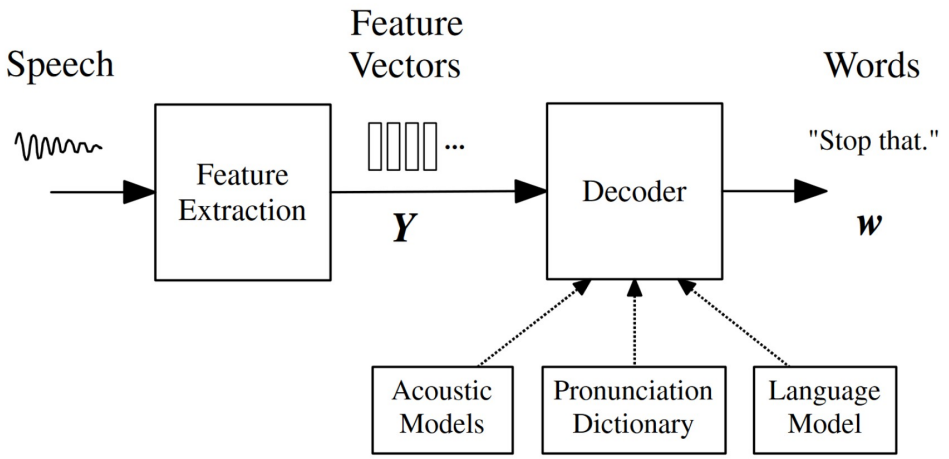


Abbildung 4: Die Architektur eines auf HMM basierenden Spracherkennungssystems. Quelle: [38, S. 201]

2.3 Bewertung von Spracherkennungssystemen

Um die Ergebnisse eines Spracherkennungssystems zu bewerten, existieren verschiedene Metriken. Die geläufigste ist dabei die *Word Error Rate* (*WER*, *Wortfehlerrate*), welche den prozentualen Anteil an Fehlern im erkannten Text angibt [48, S. 2]. Dabei wird das Ergebnis mithilfe des Referenztextes hinsichtlich drei Fehlerarten analysiert:

- Auslassung / Löschung (Deletion, D): Das Wort wurde vom Spracherkennungssystem ausgelassen.
- Ersetzung (Substitution, S): Das Wort wurde durch ein anderes Wort ersetzt.
- Einfügung (Insertion, I): Ein nicht gesprochenes Wort wurde zusätzlich eingefügt. [48, S. 2], [101, S. 334-335], [29, S. 21-23]

Ein einfaches Beispiel dieser Fehlerarten ist in Tabelle 4 zu sehen.

Referenz:	—-	die	Katze	lief	im	Schnee
Erkannt:	und	sie	Katze	lief	—-	Schnee
Fehler:	I	S			D	

Tabelle 4: Beispiel der Fehlerarten. Quelle: [48, S. 2]

Die Word Error Rate (WER) lässt sich schließlich anhand der Anzahl der jeweiligen Fehler sowie der Anzahl der Wörter, wie in Formel 1 gezeigt, berechnen:

$$WER = \frac{S+D+I}{N} \tag{1}$$

Dabei ist:

S die Anzahl der Ersetzungen

D die Anzahl der Auslassungen

I die Anzahl der Einfügung

N die Anzahl der Wörter im Referenztext [48, S. 2]

Je nach Ausrichtung der Texte unterscheiden sich auch gegebenenfalls die Anzahl der Fehler. Deshalb kann die minimale Anzahl der Fehler von einer Referenzwortfolge $R = r_1, r_2, ...r_n$ und der Wortfolge des Erkenners $H = h_1, h_2, ...h_m$ mittels Formel 2 berechnet werden. Man nennt dies auch die „minimale Editierdistanz“ [101, S. 335]. [101, S. 335]

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 \\ D(i-1,j-1)+d(i,j) \\ D(i,j-1)+1 \end{cases}$$

Dabei ist:

$$d(i,j) = \begin{cases} 1 \text{ falls } r_i \neq h_j \\ 0 \end{cases} \tag{2}$$

mit:

$$D(i,j) = \infty \text{ für } i < 1 \vee j < 1$$

mit der Ausnahme:

$$D(0,0) = 0 \text{ [101, S. 335]}$$

Daraus ergibt sich aus 1 die folgende angepasste Formel (3):

$$WER = \frac{D(N,M)}{N}$$

Dabei ist: (3)

D die minimale Editierdistanz

N die Anzahl der Wörter im Referenztext

M die Anzahl der Wörter im erkannten Text [48, S. 2], [101, S. 335]

Es wird häufig an der Word Error Rate kritisiert, dass keine Unterscheidung zwischen informativen und nicht-informativen Wörtern vorgenommen wird. So spielt es für die Word Error Rate keine Rolle, ob es sich bei dem Fehler um ein informatives Wort oder beispielsweise einen Artikel handelt. Da diese Betrachtung die Komplexität der Metrik deutlich erhöhen würde, hat sich eine solche noch nicht durchgesetzt. [48, S. 2]

3 DeepSpeech

DeepSpeech ist eine von Mozilla entwickelte Open Source Speech-To-Text-Engine, welche mittels verschiedener Machine Learning Techniken die Transkription von Audiomaterial ermöglicht. Ziel ist und war es ein System zu entwickeln,

- welches zur Ausführung keine hohen Hardwareanforderungen benötigt.
- dessen Code und Modelle unter der offenen Mozilla Public Licence veröffentlicht werden.
- welches von vielen verschiedenen Plattformen und Programmiersprachen unterstützt wird. [70], [76]

Im Gegensatz zu herkömmlichen Spracherkennungssystemen (Kapitel 2) durchlaufen die Merkmale des Audiomaterials in DeepSpeech ein tiefes rekurrentes neurales Netz (engl. *deep recurrent neural network*). Die Ausgabe dieses Netzes ist schließlich die aus dem Material resultierende Transkription. Ursprünglich basiert DeepSpeech auf der in dem Paper „*Deep Speech: Scaling up end-to-end speech recognition*“ [47] beschriebenen DeepSpeech Engine von Baidu. Die beiden Engines unterscheiden sich jedoch heutzutage an einigen Stellen. [64], [76], [70]

3.1 Gründe für die Entwicklung

Für die Entwicklung von DeepSpeech gab es für Mozilla verschiedene Gründe. Diese sollen in den folgenden Kapiteln näher beleuchtet werden.

3.1.1 Offene Alternative

Vor der Entwicklung von DeepSpeech gab es auch schon einige große Anbieter von Spracherkennungssystemen, welche qualitativ hochwertige Transkriptionen erzeugen konnten. Beispielsweise *Google Cloud Speech-To-Text* [11] oder auch *IBM Watson Speech to Text* [50]. Jedoch existieren auch heute noch kaum universell einsetzbare Open Source Spracherkennungssysteme. Die Existierenden unterstützen in der Regel nur vereinzelte Sprachen. Ziel von Mozilla war es also, eine offene Lösung für jeden zu entwickeln, die viele unterschiedliche Sprachen unterstützt. Dabei soll besonders die Wirtschaftlichkeit der einzelnen Nutzer keine Rolle spielen. [56], [17], [65], [16], [1, S. 1-2]

Ein großes Problem von Spracherkennungssystemen sind die verfügbaren Trainingsdaten. So sind einerseits öffentlich verfügbare Datensätze häufig sehr eingeschränkt und andererseits

kommerzielle Datensätze sehr kostenintensiv. Deshalb hat Mozilla 2017 zusätzlich zu DeepSpeech das Projekt „Common-Voice“ gestartet. Mit Common-Voice möchte Mozilla das Problem der Datensätze mittels Crowdsourcing lösen. Über die Website [67] können Freiwillige verschiedene Sätze in ihrer Sprache aufnehmen und bereits eingesprochene Sätze verifizieren. Wie in Abbildung 5 zu sehen ist, ergeben sich so Datensätze für eine Vielzahl von Sprachen. Der daraus resultierende Datensatz kann schließlich von jedem heruntergeladen und verwendet werden. Zusätzlich sammelt Mozilla an dieser Stelle auch noch weitere Quellen zu öffentlichen Datensätzen. [16], [67], [68], [69], [17]

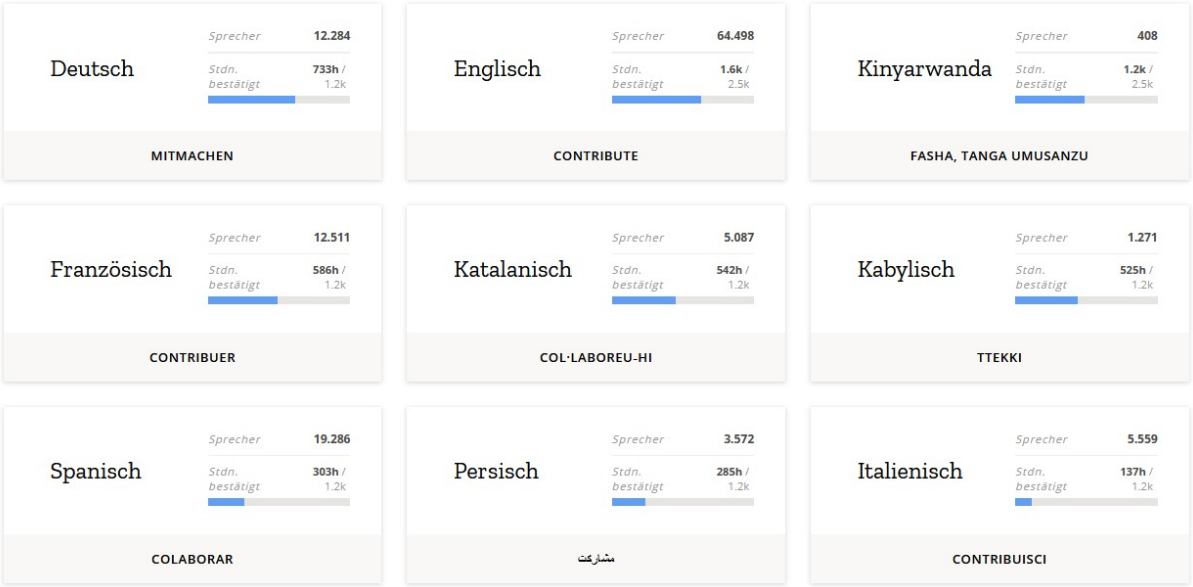


Abbildung 5: Ausschnitt der bereits unterstützten Sprachen auf Common-Voice.
Quelle: [68], (Stand: 23.09.2020)

3.1.2 Offline Support

Hinsichtlich der Privatsphäre sind die bisher genannten Dienste, wie *Google Cloud Speech-To-Text* oder auch *IBM Watson Speech to Text* und andere, bedenklich. So bieten diese häufig nur einen Web-Service an, an den das zu transkribierende Material gesendet werden muss. Je nach Einsatzzweck könnte dies zu Verstößen gegen gesetzliche Regelung oder etwaigen Richtlinien führen. Hinzu kommt, dass die Web-Services gegebenenfalls über ein Datenlimit verfügen und durch eine schlechte Replizierbarkeit nicht für die Forschung eingesetzt werden können. Ziel war es daher eine Lösung zu entwickeln, welche offline verwendet werden kann. [1, S. 1], [56], [17], [11], [50]

3.1.3 Machine Learning Ansatz

Wie in Abbildung 6 zu sehen ist, ist das zukünftige Ziel von DeepSpeech die bisherigen Komponenten eines Spracherkennungssystems vollständig durch ein neurales Netz zu ersetzen. Durch den Machine Learning Ansatz hatte Mozilla das Ziel, den Programmieraufwand insgesamt zu verringern. So entsteht durch den Machine Learning Ansatz kein zusätzlicher Code um weitere Sprachen zu unterstützten. Es muss „lediglich“ das Modell der Sprache trainiert werden. Ebenso sorgt der erhöhte Abstraktionsgrad durch das Machine Learning zu einer erhöhten Qualität der Ergebnisse. [56], [47]

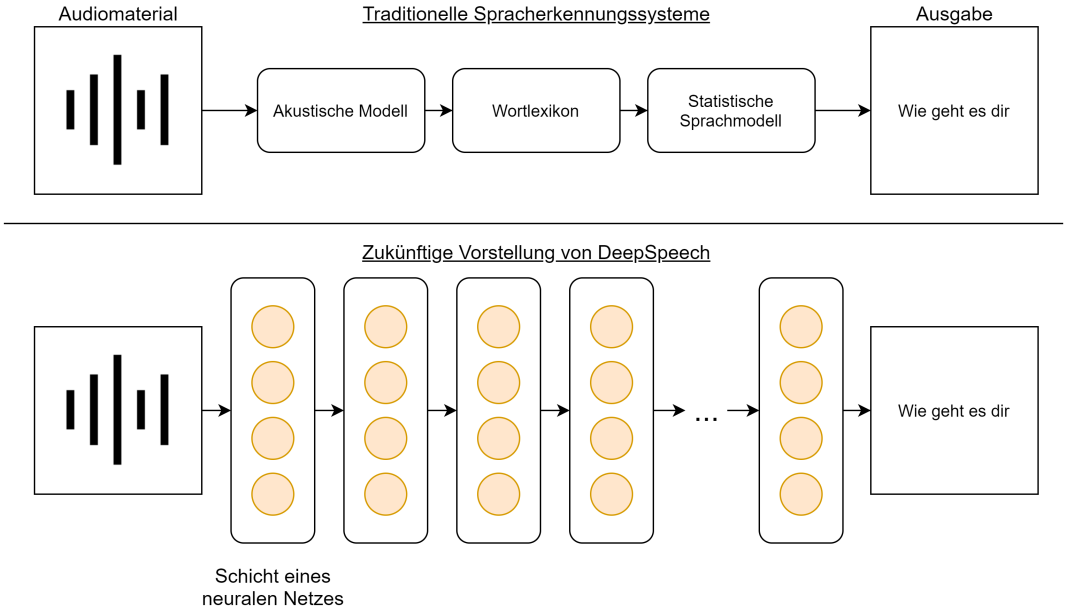


Abbildung 6: Traditionelle Spracherkennungsysteme im Vergleich zur Vorstellung von DeepSpeech. Angepasste Darstellung von: [46, 47-48min]

3.2 Gründe für die Verwendung

Wie in Kapiel 3.1.1 dargelegt wurde, existieren nach [56] nur sehr wenige offene Alternativen zu kommerziellen Spracherkennungssystemen. Trotzdem soll an dieser Stelle durch einen kleinen Vergleich zwischen DeepSpeech und einer Handvoll anderer Open Source Speech-To-Text-Engines die Verwendung von DeepSpeech begründet werden.

Zwei der bekanntesten Speech-To-Text-Engines sind Julius und Kaldi. Beide basieren dabei auf einem Ansatz mittels HMM, wie er auch schon in Kapitel 2 erläutert wurde. Während DeepSpeech sich sehr leicht mittels `pip` installieren lässt, müssen für Julius und Kaldi die jeweiligen Git-Repositories gebaut / kompiliert werden. Neben einem englischen Modell wird von Julius ebenso ein japanisches zur Verfügung gestellt. Andere Sprachen werden von dem Voxforge

Projekt unterstützt. Fast alle Modelle sind jedoch mehrere Jahre alt oder basieren auf einem sehr kleinen Datensatz. Das deutsche Modell von Voxforge besteht beispielsweise aus insgesamt 57h Material und ist damit eines der Größten (Stand 28.09.2020). [96], [54], [55], [104], [76]

Kaldi hingegen, veröffentlicht neben verschiedenen englischen Modellen auch einige in Mandarin. Neben einem sehr guten deutschen Modell, bereitgestellt von der Universität Hamburg, ist es hier jedoch schwer, weitere Modelle in anderen Sprachen zu finden. Im Vergleich dazu, lassen sich für DeepSpeech schnell verschiedene Modelle finden. So stellt zum Beispiel das DeepSpeech-Polyglot Projekt, Modelle für Deutsch, Spanisch, Französisch, Italienisch und Polnisch zur Verfügung. Es existieren aber auch Modelle für Walisisch oder Kabylisch. Das Trainingsmaterial liefert dabei zum größten Teil das Common Voice Projekt. Je nach Sprache werden für die Modelle zwischen 80h (Walisisch) bis 1.500h (Deutsch) Material verwendet. [55], [54], [62], [103], [53]

Eine weitere bekannte Alternative ist CMUSphenix. Die Entwickler haben jedoch in einem Blogpost 2019 bekannt gegeben, dass sie größtenteils an anderen Projekten mitwirken. Eines dieser Projekte ist dabei die Speech-To-Text-Engine Vosk, welche auf das bereits vorgestellte Kaldi aufsetzt. Die Installation von Vosk funktioniert ebenso wie bei DeepSpeech über pip. Es werden direkt Modelle in verschiedenen Größen (MB bis GB) für 13 verschiedenen Sprachen bereitgestellt. Die WER (Kapitel 2.3) für die englische Sprache liegt dabei je nach Modell zwischen 7.08% und 15.34%. Im Vergleich dazu, liegt die aktuelle englische WER von DeepSpeech bei 5,97%, mit einer Modellgröße von ca. 180 MB. Die anderen Modelle von Vosk sind aktuell (Stand 28.09.2020) noch ohne veröffentlichte WER. [74], [9], [8], [3]

Wie auch DeepSpeech verfolgen die Speech-To-Text-Modelle von Silero für PyTorch ebenso einen Machine Learning Ansatz. Dabei erzielen diese, besonders in Hörbüchern und Erzählungen, eine gute bis sehr gute WER. Die Modelle sind in einer frei verfügbaren Community Edition sowie in einer kostenpflichtigen Enterprise Edition verfügbar. Die Enterprise Modelle sind dabei (offensichtlich) immer einige Prozente besser. Im Gegensatz dazu ist das von Mozilla veröffentlichte Modell unter der offenen Mozilla Public License 2.0 verfügbar. Wie in Kapitel 3.1.1 erläutert, verfolgt Mozilla mit DeepSpeech einen klar freien und offenen Ansatz. [74], [100], [99, Releases]

Insgesamt bietet DeepSpeech also folgende Vorteile:

- Einfache Installation mittels pip.
- Die Community stellt eine Vielzahl von Modellen für unterschiedliche Sprachen zur Verfügung.
- Modelle benötigen wenig Speicherplatz.

- Von Mozilla bereitgestellte Komponenten und Daten werden unter der offenen Mozilla Public License 2.0 veröffentlicht.
- Wenig Code und Overhead für die Verwendung nötig.

3.3 Wie funktioniert DeepSpeech?

Das Herzstück von DeepSpeech ist ein rekurrentes neurales Netz. Rekurrente neurale Netze erlauben es Informationen zu speichern und diese als zusätzlichen Input zu verwenden. Dies ist besonders bei der Verarbeitung von sequenziellen Informationen wie Audio oder Text wichtig. Man nennt diese Art von Netze daher auch „Sequence-to-Sequence Netze“ [63]. [66], [63]

Die Transkription mittels DeepSpeech (siehe Abbildung 7) beginnt ähnlich wie die der traditionellen Spracherkennungssysteme. Zuerst wird das Audiomaterial in kleinere, sich überlappende Stücke geschnitten. Aus diesen Stücken werden schließlich die Merkmale, darunter das jeweilige Spektrogramm, extrahiert. [66], [64]

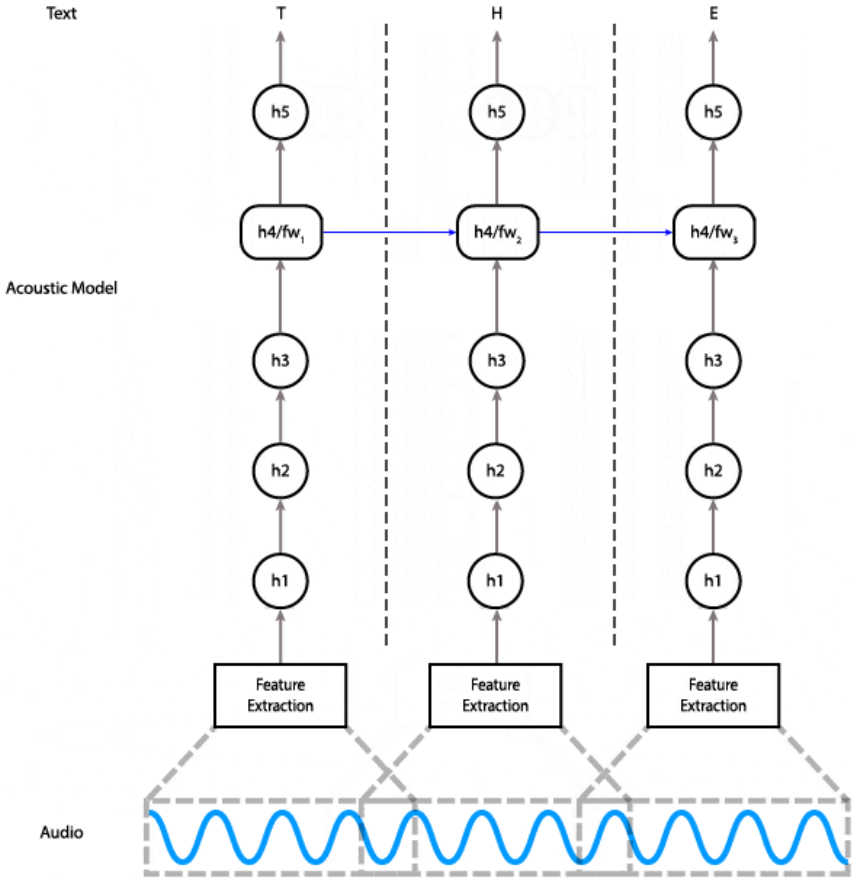


Abbildung 7: DeepSpeech Architektur. Quelle: [66]

Danach durchlaufen diese Merkmale die ersten drei ($h1$-$h3$) voll verbundenen Schichten (engl. *fully connected layers*) des neuralen Netzes. In diesen Schichten sind die Neuronen mit allen folgenden Neuronen verbunden und beginnen damit die extrahierten Daten zu analysieren. Das bisherige Ergebnis durchläuft dann eine rekurrente Schicht ($h4$), welche es ermöglicht, das Ergebnis einerseits an die nächste voll verbundene Schicht ($h5$) und andererseits an die rekurrente Schicht der folgenden Auswertung ($h4/fw_2$ oder $h4/fw_3$) weiterzugeben. So kann das vorherige Ergebnis in der folgenden Auswertung mit einbezogen werden. [66], [64], [63], [95], [1, S. 2]

Die Ausgabe des neuralen Netzes entspricht dabei einer Matrix (Abbildung 8), welche für die einzelnen Abschnitte die Wahrscheinlichkeiten für jeden Buchstaben enthält. Hier wird auch ein großer Unterschied zu herkömmlichen Spracherkennungssystemen deutlich. Während diese versuchen Phone zu erkennen, erkennt DeepSpeech tatsächliche Zeichen wie Buchstaben, Leerzeichen und Apostrophe. [64], [70], [1, S. 2]

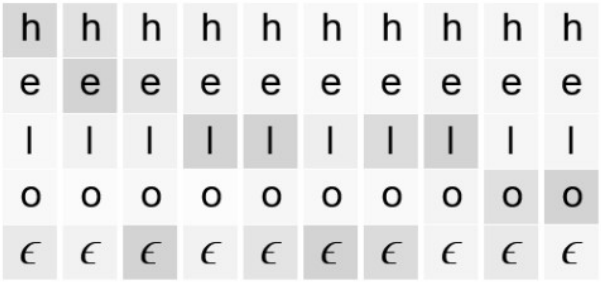


Abbildung 8: Beispiel Ergebnis des neuralen Netzes. Je intensiver der Grauton, desto höher die Wahrscheinlichkeit. Quelle: [45]

Die resultierende Matrix wird schließlich mithilfe des Connectionist Temporal Classification-Algorithmus (CTC) zu der richtigen Ausgabe umgewandelt. Der große Vorteil des CTC-Algorithmus ist, dass für das Training keine Ausrichtung des Textes auf das Gesprochene nötig ist. Die jeweilige Transkription muss also nicht mit Zeitstempel für jedes Zeichen versehen werden. Im ersten Schritt des CTC-Algorithmus werden die Zeichen nach ihrer Wahrscheinlichkeit absteigend zu mehreren möglichen Sequenzen zusammengefasst (Abbildung 9). Aufgrund der überlappenden Abschnitte kann es vorkommen, dass ein Zeichen mehrfach hintereinander auftaucht. Deshalb werden im nächsten Schritt doppelte Zeichen, wie in Abbildung 10 zu sehen ist, zusammengefasst. Um Wörter mit Doppelkonsonanten trotzdem richtig zusammenzufassen, wurde das ε-Zeichen beim Training hinzugefügt. Dieses symbolisiert zwischen zwei gleichen Konsonanten, dass es sich hierbei um einen Doppelkonstanten handelt und verhindert somit, dass diese zusammengefasst werden. Um die möglichen Transkriptionen (Abbildung 11) zu erhalten, wird das ε-Zeichen im dritten Schritt ebenfalls entfernt. [45], [64]

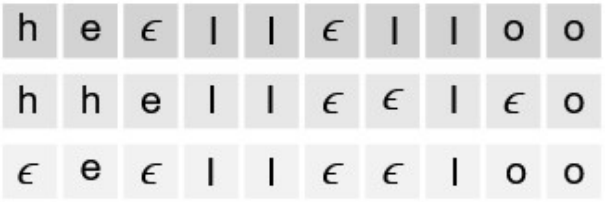


Abbildung 9: Beispiel Sequenzen aus der Matrix. Quelle: [45]

Abbildung 10: Beispiel Anwendung des CTC-Algorithmus. Quelle: [45]

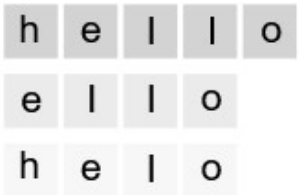


Abbildung 11: Mögliche Ergebnis-Transkriptionen. Quelle: [45]

Problematisch ist allerdings, dass auch DeepSpeech bis hierhin ein Problem mit Homophonen wie beispielsweise „mehr“ und „Meer“ hat. Deshalb wurde auch in DeepSpeech als letzter Schritt ein Sprachmodell eingebunden, welches die Wahrscheinlichkeit der Wortfolgen untersucht und bewertet. Dies hat die *Word Error Rate* des englischen Modells von 16% auf fast menschliche 6,5% verringert. [45], [64]

3.4 Beispiel Verwendung

Um DeepSpeech einsetzen zu können, muss es zuerst mittels pip, wie im Programmausdruck 1 zu sehen ist, installiert werden. Im Projekt wurde die Version 0.9.2 verwendet.

```
1 pip install deepspeech
```

Programmausdruck 1: Installation von DeepSpeech mittels pip

Die eigentliche Verwendung ist in Programmausdruck 2 zu sehen. Zuerst (Z. 1-3) wird `deepspeech`, `wave` und `numpy` importiert. Danach (Z. 6) muss das Model initialisiert werden. Dafür wird einfach der Pfad zur Model-Datei dem Konstruktor übergeben. In der darauffolgenden Zeile wird der Scorer (das Sprachmodell) aktiviert und gesetzt. Die zu transkribierende wav-Datei wird daraufhin mittels wave geöffnet (Z. 9) und durch numpy in ein 16 Bit int-Array konvertiert (Z. 11-12). Zuletzt muss das resultierende Array nur noch der `stt`-Methode übergeben werden. Das zurückgegebene Ergebnis (Z. 14) enthält den kleingeschriebenen und keine Satzzeichen enthaltenden transkribierten Text. [43], [75]

```
1  import deepspeech
2  import wave
3  import numpy as np
4  ...
5  # Init DeepSpeech
6  model = deepspeech.Model(model_path)
7  model.enableExternalScorer(scorer_path)
8  # Open File
9  w = wave.open(filename)
10 # Convert to 16-bit int array
11 buffer = w.readframes(w.getnframes())
12 data = np.frombuffer(buffer, dtype=np.int16)
13 # Get Result
14 result = model.stt(data)
```

Programmausdruck 2: Verwendung von DeepSpeech unter Python, angelehnt an [43], [75]

Da DeepSpeech mit Audiomaterial trainiert wird, welches nur ein bis zwei Sätze beinhaltet, funktioniert die Transkription am besten, wenn das Material eine ähnliche Länge besitzt. Um längeres Material zu transkribieren, sollte es also in kürzere Segmente zerteilt werden. Dazu wird in den angegebenen Beispielen `transcribe.py` [74] sowie `wavTranscriber.py` [72] der `WebRTC Voice Activity Detector` (VAD) eingesetzt. Hierbei wird das Material erst in 30ms kleine Abschnitte zerteilt und mittels des WebRTC VAD überprüft, ob im Material gesprochen wurde. Die gefundenen Abschnitte, in denen gesprochen wurde, werden dann wie oben gezeigt ausgewertet und schließlich zusammengeführt. Durch die Verwendung des WebRTC VAD hat sich eine deutlich bessere WER ergeben.

4 Konzeption des Editors

Im Folgenden wird die Konzeption des Editors beschrieben. Hierzu werden zuerst mögliche Ideen und Anforderungen in einem Brainstorming gesammelt. Im Anschluss folgt ein Test verschiedener Transkriptionsprogramme, um auch hier etwaige Anforderungen und Schwachstellen zu identifizieren.

Auf Basis der gesammelten Erkenntnisse werden danach die Anforderungen nach der Anforderungsschablone von Chris Rupp [94] definiert. Es folgen die ersten Wireframes sowie die Architekturbeschreibung.

4.1 Brainstorming

Um möglichst viele kreative Ideen und Anforderungen zu sammeln, wurde zuerst ein Brainstorming durchgeführt. Die Ergebnisse lassen sich in Abbildung 12 finden. Eine größere Version desselben Bildes befindet sich aus Gründen der besseren Lesbarkeit nochmals im Anhang (Abbildung 60).

Abbildung 12: Brainstorming: Ideen und Anforderungen

Wie zu sehen ist, sind die Ideen und Anforderungen in drei Kategorien unterteilt. Zu den *Grundfunktionen* gehören alle Funktionen, die der Editor auf jeden Fall bereitstellen muss. Dazu gehören sowohl das Öffnen, Bearbeiten und Speichern von Transkriptionen, als auch die Möglichkeit, Audio- und Videomaterial abzuspielen. Der Editor soll außerdem die Möglichkeit bieten, die Schriftgröße sowie die Abspielgeschwindigkeit anzupassen. Sofern die verschiedenen Sprachmodelle beim Nutzer verfügbar sind, sollen ebenso unterschiedliche Sprachen unterstützt werden.

Die Kategorie *Aufbereitung* beschreibt Möglichkeiten, den von DeepSpeech transkribierten Text aufzubereiten. Der wichtigste Punkt darunter ist die Rechtschreibung. Dazu zählt besonders die Wiederherstellung von Groß- und Kleinschreibung und die der Satzzeichen. Des Weiteren könnte der Text mittels alternativer Speech-To-Text-Engines, wie das bereits vorgestellte Vosk, um alternative Wortvorschläge ergänzt werden. Damit etwaige wiederkehrende Änderungen schnell durchgeführt werden können, sollte es die Möglichkeit geben, den Text mittels vordefinierter Regelketten zu bearbeiten. Um den Text mit Zeitstempeln zu versehen, könnten verschiedene *Forced Alignment*-Algorithmen zum Einsatz kommen. Damit Füllwörter oder ungewollte Mitschnitte nicht ebenfalls transkribiert werden, sollte es die Möglichkeit geben, diese auszuschneiden.

In der Kategorie *Weiterverarbeitung* wurden Ideen gesammelt, um das Ergebnis weiterzuverarbeiten. Um die wichtigsten Fakten einer Transkription zu erhalten, könnte beispielsweise eine automatische Zusammenfassung generiert werden. Sollte es sich bei der Transkription um ein Gespräch handeln, könnte eine automatische Sprechererkennung zum Einsatz kommen, um die jeweiligen Sprecher im Text zu hinterlegen. Ist das Ziel den Text an andere Personen weiterzugeben, so bietet sich eine Anonymisierung und Untersuchung der Lesbarkeit des Textes an. Handelt es sich bei dem Ausgangsmaterial um ein Video, so könnte das Ergebnis als Untertitel-Datei (z.B. *.srt* oder *.ass*) exportiert werden.

4.2 Test anderer Transkriptionsprogramme

Um klassische Anforderungen und Schwächen von Transkriptionsprogrammen zu entdecken, wird nachfolgend eine Handvoll dieser genauer untersucht.

4.2.1 easytranscript

Um easytranscript [106] zu verwenden, muss JAVA sowie der VLC-Player installiert sein. Damit eine Transkription durchgeführt werden kann, muss der Nutzer zuerst ein Projekt erstellen. Dazu muss der Projektname, der Name des ersten Transkriptions-Dokumentes und die dazugehörige Mediendatei angegeben werden. Es erscheint die in Abbildung 13 zu sehende Maske.

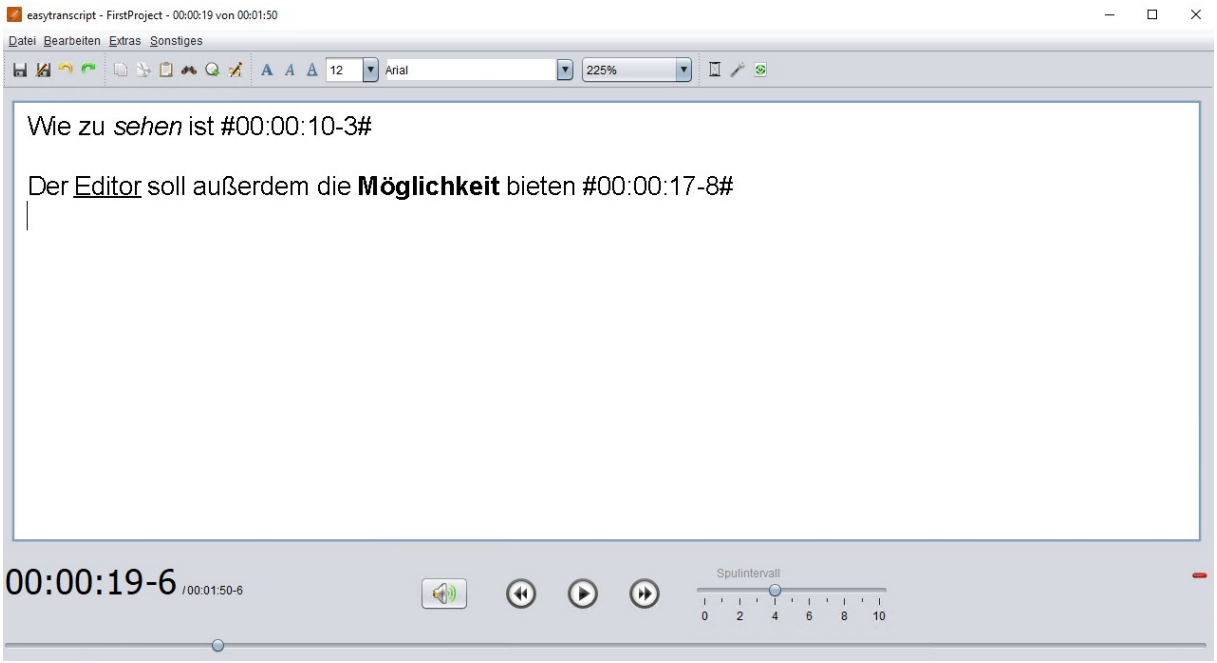


Abbildung 13: easytranscript: Hauptmaske

Neben klassischen Operationen wie Speichern, Rückgängig, Kopieren und Einfügen, beinhaltet die Toolbar ebenso die Möglichkeit, die Schriftart, die Schriftgröße und die Schriftauszeichnung (**Fett**, *Kursiv*, Unterstrichen) anzupassen. Realisiert wird dies durch die Verwendung des Rich Text Formats (RTF). Zusätzlich kann über die Toolbar ein Zeitstempel eingefügt und der Sprecherwechsel beim Betätigen der Entertaste aktiviert werden. Die Funktionen in der Toolbar lassen sich ebenso mittels Tastenkombination ausführen. Leider werden diese nur in der Online-Hilfe angezeigt und können nicht verändert werden.

Über die Einstellungen ist es aber möglich, Textbausteine und automatisch zu ersetzende Kürzel zu definieren. Im unteren Bereich des Editors befindet sich die Steuerung des Ausgangsmaterials. Das Ausgangsmaterial lässt sich nicht nur mit den sichtbaren Buttons, sondern auch mittels der Funktionstasten (F3 bis F5) steuern. Dabei kann sowohl das Spulintervall als auch die Geschwindigkeit und Lautstärke angepasst werden.

4.2.2 oTranscribe

Bei oTranscribe [82] handelt es sich um ein Transkriptionsprogramm, welches im Browser ausgeführt wird. Dabei kann entweder die Web-Version unter https://otranscribe.com/ verwendet oder der Code aus dem GitHub-Repository geklont und mittels Node ausgeführt werden. Um die Transkription zu beginnen, muss zuerst das Ausgangsmaterial hochgeladen werden. In der Hilfe wird darauf hingewiesen, dass die Datei im lokalen Browser verbleibt und den eigenen Computer nicht verlässt [83]. Es erscheint die in Abbildung 14 zu sehende Maske.

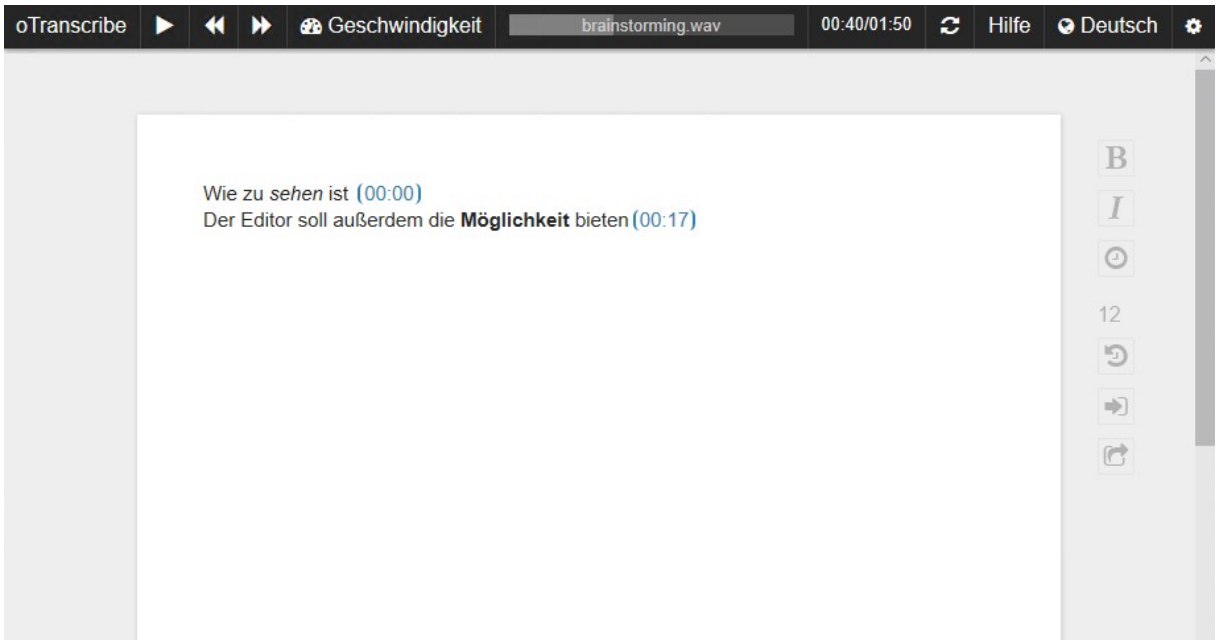


Abbildung 14: oTranscribe: Hauptmaske

In der Toolbar kann das Material gesteuert und die Geschwindigkeit angepasst werden. Über die Seitenleiste lässt sich die Schriftauszeichnung (**Fett**, *Kursiv*) anpassen und ein Zeitstempel einfügen. Darunter wird die Anzahl der Wörter angezeigt und die Möglichkeit gegeben, eine Historie einzusehen sowie das Dokument zu exportieren oder zu importieren. Der Export erfolgt wahlweise als Markdown, einfachen Text (*.txt*-Datei) oder im oTranscribe-Format. Bis auf die Historie sowie den Export und Import können alle genannten Funktionen ebenfalls über eigene oder vordefinierte Tastenkombinationen ausgeführt werden.

4.2.3 f4transkript

Für f4transkript [5] müssen ebenso wie bei dem bereits vorgestellten easytranscript JAVA und der VLC-Player installiert sein. Da sich easytranscript an f4transkript orientiert [102], ist der Funktionsumfang der beiden Transkriptionsprogramme recht ähnlich. So können wie in Abbildung 15 zu sehen ist, die Schriftart und die Schriftauszeichnung in der Toolbar ebenfalls angepasst werden. Im rechten Bereich der Maske können zusätzlich Kommentare festgehalten, eine Statistik zur Bearbeitungsdauer eingesehen und verschiedene Textbausteine verwaltet werden.

Abbildung 15: f4transkript: Hauptmaske

Im unteren Bereich kann die Audiospur abgespielt und die Lautstärke, die Geschwindigkeit und das Spulintervall angepasst werden. Diese und andere Funktionen, wie beispielsweise das Setzen von Zeitstempel, sind per anpassbare Tastenkombinationen ebenfalls ausführbar. In der kostenlosen Version von f4transcript ist das Abspielen des Ausgangsmaterials auf jeweils fünf Minuten begrenzt.

4.2.4 autoEdit

In autoEdit [85] werden mehrere Transkriptionen in einem Projekt verwaltet. Bevor allerdings die Transkription durchgeführt werden kann, wird das Ausgangsmaterial zuerst automatisch transkribiert. Dazu verwendet autoEdit entweder DeepSpeech, AssemblyAI oder Speechmatics. Für die letzteren beiden ist allerdings ein API-Key notwendig, da diese kostenpflichtig sind. Nach offizieller Dokumentation wird Speechmatics allerdings nicht mehr unterstützt. Für DeepSpeech kann über die Einstellungen ein 1.8 GB großes englisches Modell heruntergeladen werden. Aufgrund der Modellgröße ist davon auszugehen, dass es sich hierbei um eine ältere Version handelt. Abseits von Englisch ist es nicht möglich, andere Sprachen einzusetzen. [86]

Nach dem autoEdit den Text automatisch transkribiert hat, kann die in Abbildung 16 zu sehende Maske aufgerufen werden. Im oberen Bereich der Maske kann die Audiowiedergabe gesteuert werden. Rechts neben dem eigentlichen Textfeld befinden sich die Funktionen, um das Dokument in verschiedene Formate zu exportieren und zu speichern sowie die Möglichkeit, einen Paragrafen oder ein „[INAUDIBLE]“ (für unverständliche Abschnitte) einzufügen. Außerdem

kann hier aktiviert werden, dass bei Eingabe des Nutzers die Audiowiedergabe gestoppt oder dass bei einem Doppelklick auf einen Paragrafen zur Stelle im Ausgangsmaterial gesprungen wird. Eine besondere Funktion von autoEdit ist die Wiederherstellung der Zeitstempel der einzelnen Paragrafen. Jegliche Funktionen müssen jedoch mit der Maus betätigt werden. Es gibt keine Möglichkeit, Funktionen mit einer Tastenkombination auszuführen oder die Tastenkombinationen zu personalisieren oder einzusehen.

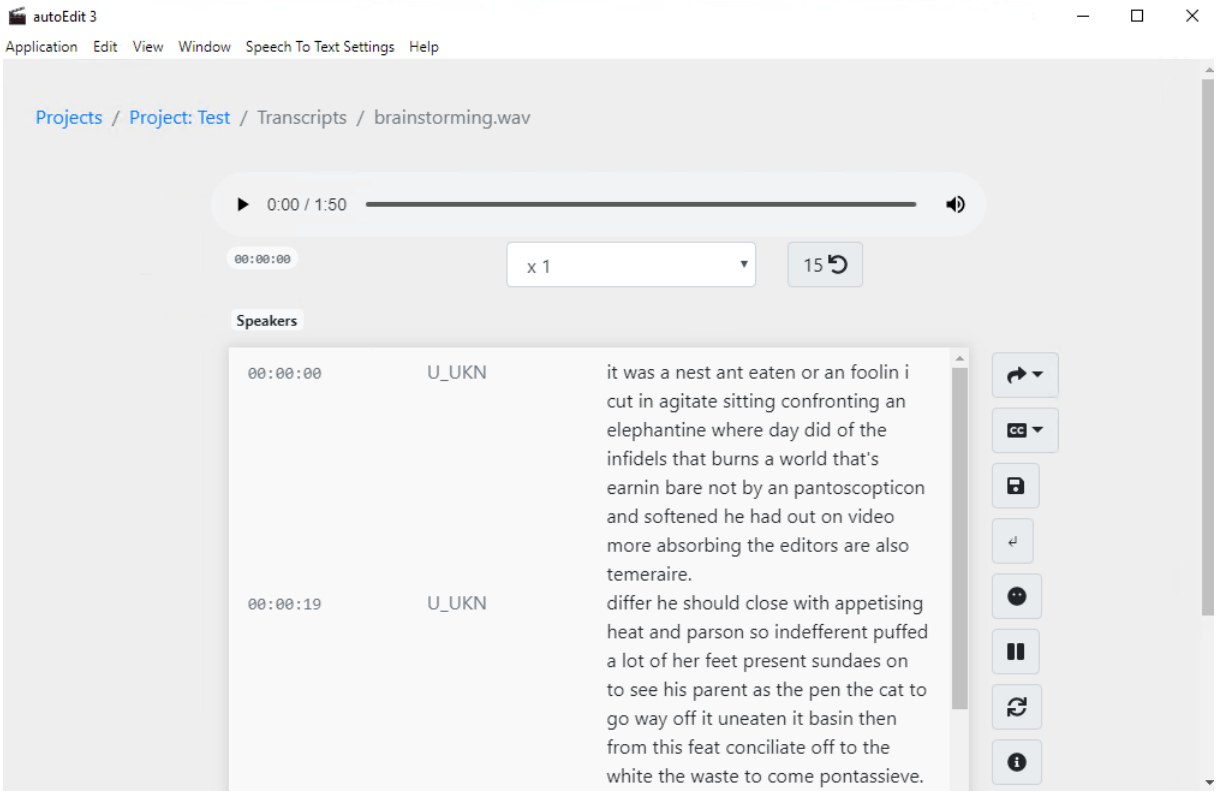


Abbildung 16: autoEdit: Hauptmaske

4.2.5 Weitere Transkriptionsmöglichkeiten

Die Transkription kann aber auch über Word, Libre Office Writer oder ähnliche Textverarbeitungsprogramme erfolgen. Dafür können beispielsweise VLC, OpenScribe oder PlayItSlowly verwendet werden, um das Abspielen der Mediendatei im Hintergrund zu übernehmen. Diese Programme erlauben unter anderem auch die Anpassung der Wiedergabe hinsichtlich der Geschwindigkeit und Lautstärke. Durch das Festlegen von Tastenkombinationen kann die Wiedergabe gesteuert werden. [102]

4.3 Erkenntnisse aus dem Test

Im Folgenden werden die Erkenntnisse aus dem Test festgehalten. Dafür erfolgt zuerst eine Wertung der Transkriptionsprogramme und anschließend eine Auflistung der gemeinsamen und einzigartigen Funktionen.

4.3.1 Wertung

Wie an den getesteten Transkriptionsprogrammen zu erkennen ist, beschäftigen sich diese hauptsächlich damit, den Nutzer während des Transkribierens zu unterstützten. Beispielsweise durch das Setzen von Zeitstempel, Einfügen von vordefinierten Textbausteinen oder der Steuerung des Ausgangsmaterials über verschiedene Tastenkombinationen. Trotzdem muss der Nutzer hier immer noch den Großteil der Arbeit selbst übernehmen. Das bedeutet, dass die Transkription häufig mindestens die Dauer des Ausgangsmaterials beansprucht.

Das vorgestellte autoEdit hingegen, stellt eine erste automatische Transkription zur Verfügung. Auch wenn dies den initialen Aufwand verringert, muss der Nutzer jedoch den Text vollständig per Hand korrigieren. Dazu zählen zum Beispiel Zeichensetzung, Groß- und Kleinschreibung, Einfügen von Umlauten und natürlich das Korrigieren von fehlenden oder gänzlich falschen Worten. Besonders in langen Transkriptionen ist dies sehr repetitiv und dadurch eine zum Teil sehr zeitaufwendige Arbeit. Hinzu kommt die fehlende Unterstützung anderer Sprachen über DeepSpeech.

4.3.2 Häufige Funktionen

Folgend werden die Funktionen aufgeführt, welche sich bei mehreren der vorgestellten Transkriptionsprogramme vorfinden lassen.

- Schriftart anpassen
- Schriftauszeichnung anpassen
- Schriftgröße anpassen
- Einfügen eines Zeitstempels
- Sprecherwechsel
- Ausführen der Funktionen mittels verschiedener Tastenkombinationen
- Möglichkeit, Textbausteine zu definieren

- Export in verschiedene Formate (Markdown, *.txt*, Untertitel, ...)
- Audiowiedergabe (Vor- und Zurückspulen, Anpassen der Geschwindigkeit und des Spulintervalls)
- Klassische (Textdatei-) Operationen (Speichern, Kopieren, Einfügen, etc.)

4.3.3 Besondere Funktionen

Folgend werden die Funktionen aufgeführt, welche nur bei wenigen der vorgestellten Transkriptionsprogramme zu finden sind.

- Kommentare
- Statistik zur Bearbeitungszeit
- Doppelklick auf den Text, um an die Stelle des Ausgangsmaterials zu springen
- Zeitstempel wiederherstellen
- Stoppen der Audiowiedergabe bei Eingabe des Nutzers
- Automatische Transkription

4.4 Anforderungen

Aus dem Brainstorming und den Tests der Transkriptionsprogramme werden nun mithilfe des FunktionsMASTeR von Rupp [94, S. 220] die Anforderungen definiert. Dabei bedeutet das Schlüsselwort „muss“, dass es sich um eine verpflichtende Anforderung handelt. Nicht verpflichtende Anforderungen werden mit dem Schlüsselwort „sollte“ versehen. [94, S. 226]

Die Grundfunktionalität wird dabei in den verpflichtenden Anforderungen festgehalten. Da sie alle umgesetzt werden müssen, sind sie nicht priorisiert. Jegliche erweiternde Funktionen, um das Ergebnis zu verbessern oder weiterzuverarbeiten, sind in den nicht verpflichtenden Anforderungen festgehalten. Eine sinnvolle Umsetzung kann zu diesem Zeitpunkt nicht gewährleistet werden. Diese sind absteigend nach eigenem Empfinden priorisiert (N00 höchste Priorität).

In den nachfolgenden Kapiteln dieser Arbeit werden die Anforderungen im Text mit ihrer jeweiligen Kennnummer (*Vxx, Nxx*) klickbar referenziert.

Nr.	**Anforderung**
V00	Das System muss dem Nutzer die Möglichkeit bieten, Transkriptionen zu speichern.
V01	Das System muss dem Nutzer die Möglichkeit bieten, Transkriptionen zu öffnen.
V02	Das System muss dem Nutzer die Möglichkeit bieten, klassische Textoperationen (kopieren, einfügen) auszuführen.
V03	Das System muss dem Nutzer die Möglichkeit bieten, eine Zeilennummerierung anzuzeigen.
V04	Das System muss dem Nutzer die Möglichkeit bieten, die Schriftgröße anzupassen.
V05	Das System muss dem Nutzer die Möglichkeit bieten, Audio- und Videomaterial abzuspielen.
V06	Das System muss dem Nutzer die Möglichkeit bieten, die Geschwindigkeit der Audio- oder Videowiedergabe anzupassen.
V07	Das System muss dem Nutzer die Möglichkeit bieten, unterschiedliche Sprachen durch die vom Nutzer bereitgestellten Sprachmodelle zu unterstützen.
V08	Das System muss dem Nutzer die Möglichkeit bieten, die verschiedenen Funktionen ebenfalls mit Tastenkombinationen ausführen zu können.
V09	Das System muss dem Nutzer die Möglichkeit bieten, Textbausteine zu definieren.
V10	Das System muss dem Nutzer die Möglichkeit bieten, die Transkription im Text-Format (*.txt*-Datei) zu exportieren.
V11	Das System muss eine erste Transkription durch DeepSpeech erzeugen.
V12	Das System muss fähig sein, den Funktionsumfang mittels Plug-ins zu erweitern.
V13	Das System muss dem Nutzer die Möglichkeit bieten, Textstellen mit Zeitstempeln zu versehen.

Tabelle 5: Verpflichtende Anforderungen

Nr.	**Anforderung**
N00	Das System sollte fähig sein, Alternativvorschläge für automatisch transkribierte Wörter anzuzeigen.
N01	Das System sollte dem Nutzer die Möglichkeit bieten, eine automatische Rechtschreibkontrolle durchzuführen.
N02	Das System sollte dem Nutzer die Möglichkeit bieten, Satzzeichen wiederherzustellen.
N03	Das System sollte dem Nutzer die Möglichkeit bieten, die Lesbarkeit der Transkription numerisch zu bewerten.
N04	Das System sollte dem Nutzer die Möglichkeit bieten, Ausschnitte aus dem Ausgangsmaterial zu entfernen.
N05	Das System sollte dem Nutzer die Möglichkeit bieten, eine gewünschte Stelle der Transkription in dem Ausgangsmaterial zu finden.
N06	Das System sollte dem Nutzer die Möglichkeit bieten, die Transkription mittels Forced Alignment mit dem Gesprochenen zu synchronisieren.
N07	Das System sollte dem Nutzer die Möglichkeit bieten, eine automatische Zusammenfassung der Transkription zu erstellen.
N08	Das System sollte dem Nutzer die Möglichkeit bieten, den Text zu anonymisieren.
N09	Das System sollte dem Nutzer die Möglichkeit bieten, die Transkription im Untertitel-Format (*.srt* oder *.ass*) zu exportieren.
N10	Das System sollte dem Nutzer die Möglichkeit bieten, eine automatische Sprechererkennung durchzuführen.

Tabelle 6: Nicht verpflichtende Anforderungen

4.5 Wireframes

Folgend die ersten Wireframes, welche den groben Aufbau der Anwendung visualisieren sollen.

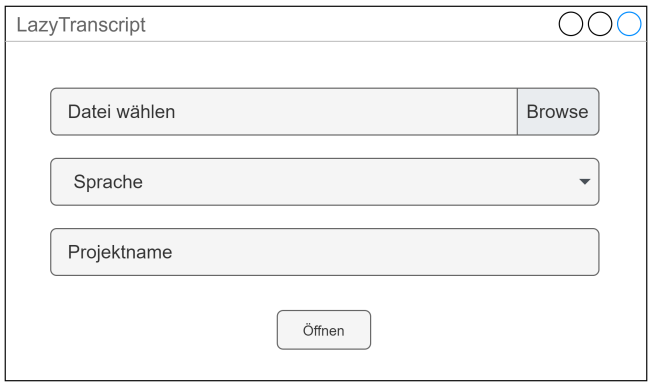


Abbildung 17: Wireframe: Eine neue Transkription erstellen oder öffnen

Der in Abbildung 17 gezeigte Dialog soll beim Öffnen der Anwendung erscheinen. Um eine neue Transkription zu erstellen, muss der Nutzer zuerst die zugrundeliegende Audio- oder Videodatei und die jeweilige Sprache des Materials auswählen. In diesem Drop-down-Menü sollen nur die Sprachen angezeigt werden, für die die jeweiligen Modelle auf dem Computer vorhanden sind. Zuletzt muss noch ein Projektname definiert werden.

Sobald die Datei geöffnet und von DeepSpeech transkribiert wurde, erscheint der in Abbildung 18 zu sehende Editor. Über die Menüleiste können klassische Operationen wie Öffnen, Speichern, Kopieren und Einfügen ausgeführt werden. Außerdem kann der Nutzer über die Menüleiste die Einstellungen sowie die Hilfe aufrufen. Einige dieser Funktionen finden sich auch in der darunterliegenden Toolbar wieder. Diese beinhaltet allerdings auch verschiedene Buttons, um Plug-ins auszuführen oder sie zu aktivieren und zu deaktivieren. In der Toolbar erhält der Nutzer ebenfalls die Möglichkeit, die Schriftart und die Schriftgröße im Editor anzupassen.

In der Mitte des Editors ist die eigentliche Transkription inklusive einer Zeilennummerierung zu sehen. Die Zeilennummerierung soll sich auf Wunsch ausblenden lassen. Darunter befindet sich die Steuerung (Play, Pause, Vor- und Zurückspulen) des Audio- oder Videomaterials. Rechts daneben kann der Nutzer die Lautstärke, die Geschwindigkeit oder auch das Spulintervall anpassen.

In Abbildung 19 sind verschiedene mögliche Fenster und Plug-ins zu sehen, welche durch die Toolbar-Buttons geöffnet werden könnten. Der Nutzer könnte sich darüber zum Beispiel eine Zusammenfassung, das jeweilige Video, die definierten Textbausteine oder den Lesbarkeitsindex [107] ansehen.

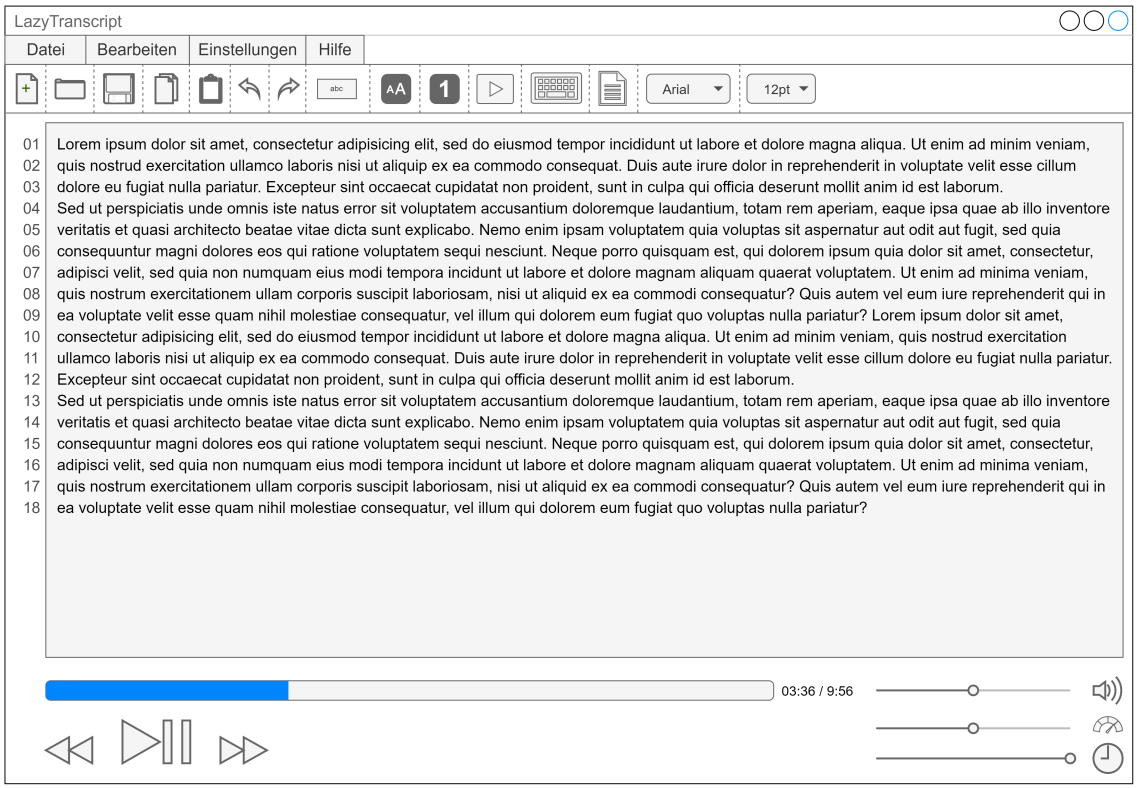


Abbildung 18: Wireframe: Editor

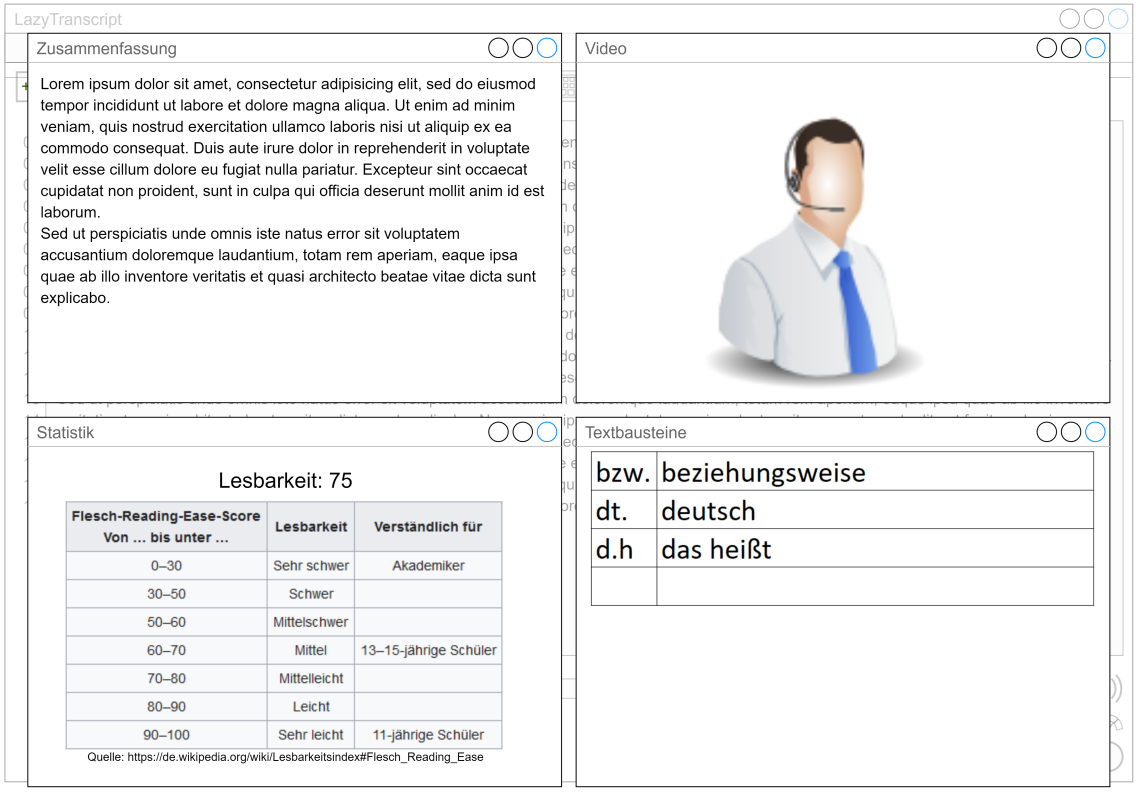


Abbildung 19: Wireframe: Zusätzliche Fenster

Für die Korrektur der Transkription, sollen die Wörter nacheinander hervorgehoben und über das Kontextmenü oder mit einem Tastaturkürzel bearbeitet werden. Wie in Abbildung 20 zu sehen ist, soll der Nutzer die Möglichkeit haben zum nächsten Wort zu springen, das Wort zu entfernen, einen Punkt zu setzen, das Wort eigenhändig umzuschreiben oder das Material an der Stelle noch einmal zu hören. Im unteren Bereich des Kontextmenüs finden sich die Vorschläge der einzelnen Plug-ins wieder. Beispielsweise die Alternativvorschläge von Vosk oder Hinweise zur Rechtschreibung.

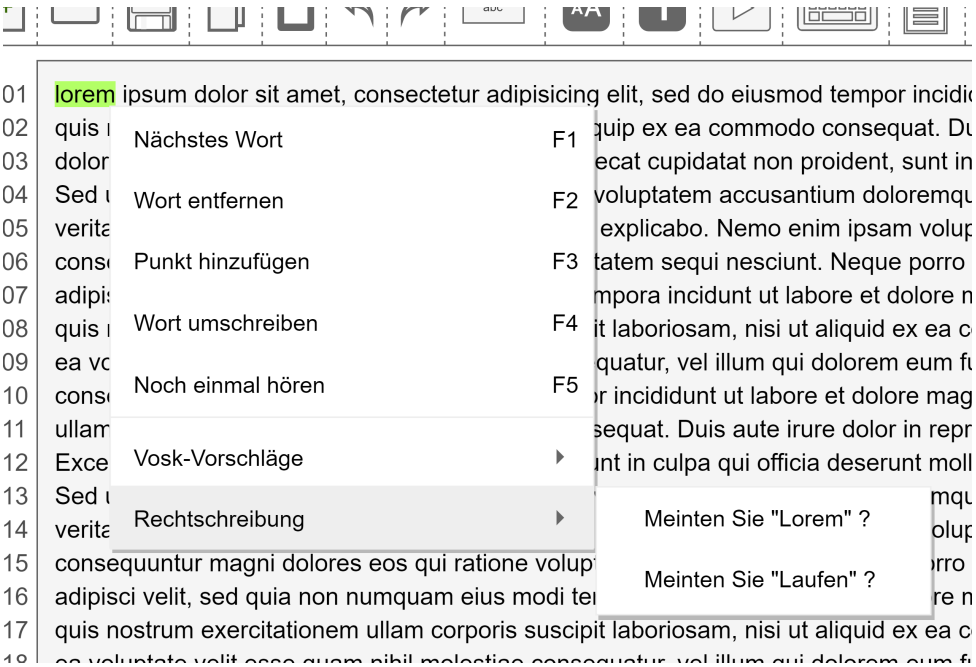


Abbildung 20: Wireframe: Korrektur

4.6 Architektur

Wie in den vorherigen Kapiteln erwähnt, soll der Editor einen Großteil des Funktionsumfangs in Form von Plug-ins erhalten. Deshalb wird im Folgenden zuerst das Plug-in-Architekturmuster und danach die gesamte Projekt-Architektur erläutert.

4.6.1 Das Plug-in-Architekturmuster

Das Plug-in-Architekturmuster beschreibt eine Architektur, bei der ein System, ohne eine Neuübersetzung, zur Laufzeit von Drittanbietern erweitert und angepasst werden kann. Realisiert wird dies durch sogenannte Plug-ins (auch Add-ons oder Add-ins genannt), welche ein bestimmtes Interface implementieren und so dem System neue Funktionen bereitstellen. [41, S. 313-314], [28, S. 132]

Um die Plug-ins zu verwalten, wird zusätzlich ein Plug-in-Manager benötigt. Dieser erzeugt zur Laufzeit die Instanzen der Plug-ins und ermöglicht der Applikation diese zu verwenden. Der Plug-in-Manager übergibt der Applikation dafür entweder direkt eine Liste der Plug-in-Objekte oder leitet den jeweiligen Aufruf an die Plug-ins weiter. Ein beispielhafter Ablauf einer Plug-in-Architektur ist in Abbildung 21 zu sehen. [41, S. 316]

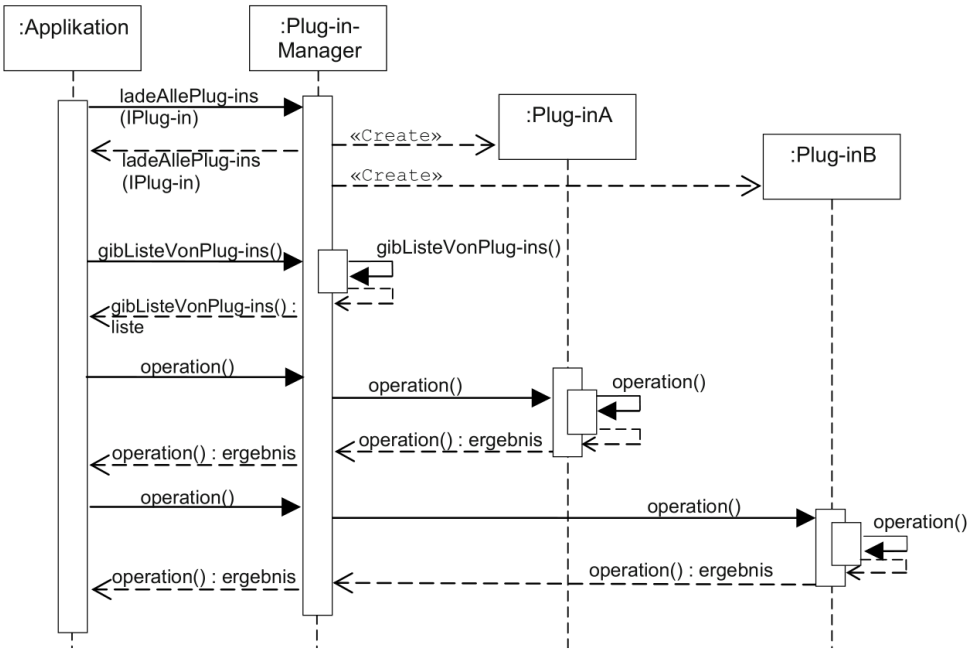


Abbildung 21: Sequenzdiagramm eines typischen Systems nach der Plug-in-Architekur. Quelle: [41, S. 317]

Eine einfache Plug-in-basierte Architektur, wie in Abbildung 22 zu sehen ist, besteht also mindestens aus

- der eigentlichen Applikation, deren Funktionsumfang erweitert werden soll.
- einem Plug-in-Manager, welcher die Plug-ins verwaltet.
- einem Interface, welches die Schnittstelle spezifiziert.
- einem Plug-in, welches das Interface implementiert und somit eine neue Funktion bereitstellt. [41, S.316-317]

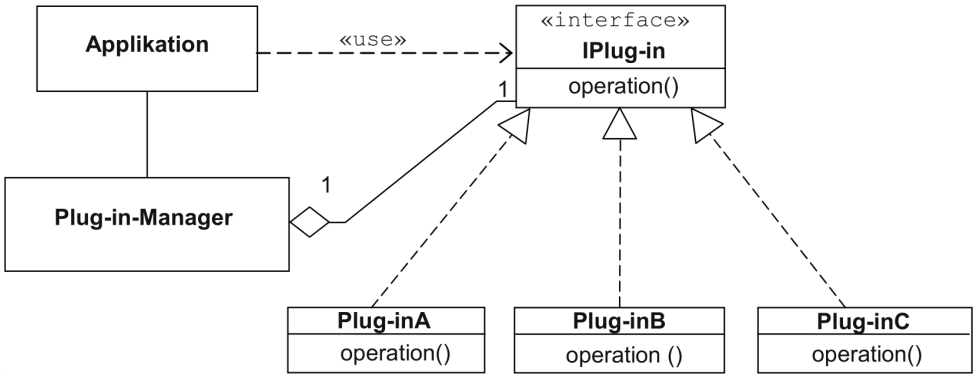


Abbildung 22: Klassendiagramm einer Plug-in-Architektur mit einem einzigen Interface. Quelle: [41, S. 316]

Das Plug-in-Architekturmuster hat dabei verschiedene Vor- und Nachteile. Ein großer Nachteil ist, dass sich sowohl der Design- als auch der Implementierungsaufwand erhöht. Dafür verantwortlich ist besonders die Definition der Plug-in-Schnittstelle(n), da Fehler hier zu starken Einschränkungen der Plug-ins führen können. Hinzu kommt die starke Abhängigkeit der Plug-ins zur Schnittstelle, weswegen diese später nur schwer verändert werden kann. Zusätzlich steigt der eigentliche Verwaltungsaufwand bei der Ausführung, da beispielsweise Fehler und Inkompatibilitäten erst zur Laufzeit entdeckt werden können. [41, S. 318], [28, S. 134-135]

Ein großer Vorteil der Plug-in-Architektur ist allerdings, dass jedes Plug-in für sich steht. Das bedeutet, dass die Plug-ins getrennt vom eigentlichen System und anderen Plug-ins entwickelt und getestet werden können. Dementsprechend sinkt natürlich auch der jeweilige Wartungsaufwand, da nur das problematische Plug-in gewartet und in einer neuen Version ausgeliefert werden muss. Hinzu kommt, dass sich Plug-in-basierte Software in der Entwicklung deutlich leichter aufteilen lässt und selbst Drittanbieter ohne Kenntnis des eigentlichen Programmcodes zusätzliche Funktionen bereitstellen können. Insgesamt führt ein Plug-in-basierter Ansatz dazu, dass die eigentliche Anwendung deutlich „schlanker“ wird, da ein Großteil der (komplexen) Funktionen ausgelagert und nur bei Bedarf geladen wird. [28, S. 133-134], [41, S. 318]

4.6.2 Übersicht

Im Folgenden wird der Architekturentwurf in Form eines Komponentendiagramms (Abbildung 23) erläutert. Eine größere Version des Komponentendiagramms ist im Anhang zu finden (Abbildung 61).

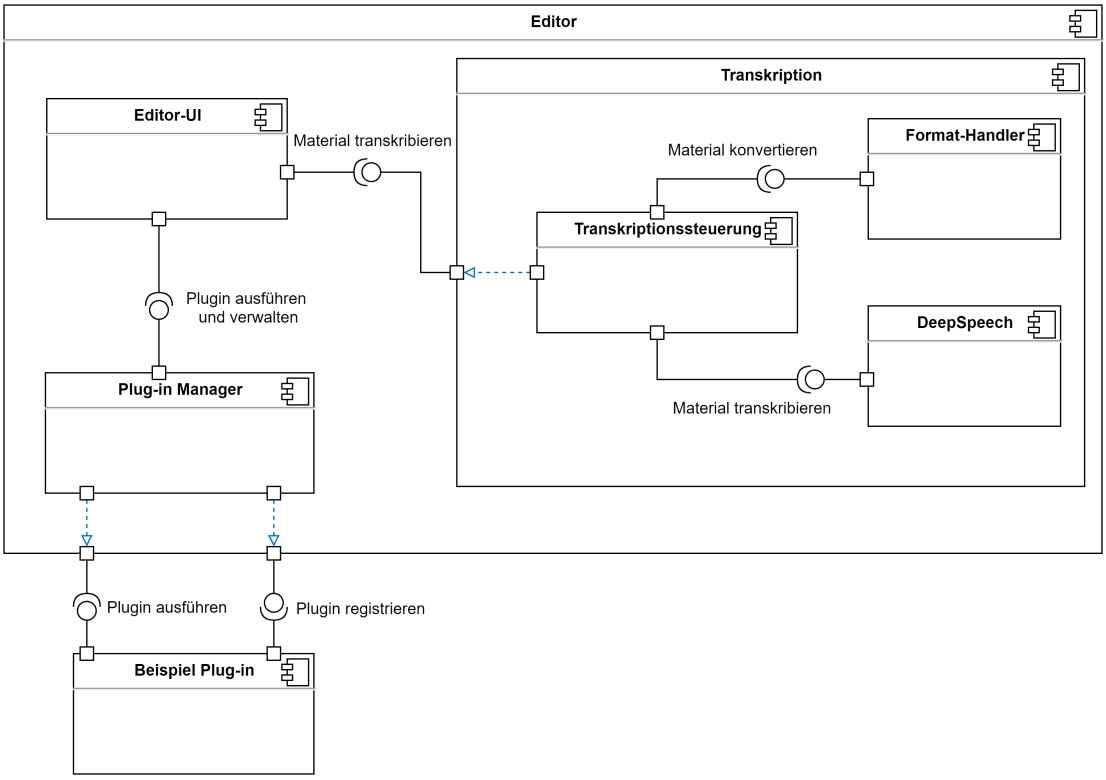


Abbildung 23: Komponentendiagramm des zu entwickelnden Editors

Der Editor soll aus drei Hauptkomponenten bestehen: Der Editor-UI, der Transkriptions-Komponente und dem Plug-in-Manager. Die Editor-UI soll für die Darstellung, wie sie in den Wireframes (Kapitel 4.5) zu sehen ist, zuständig sein. Nach Auswahl der jeweiligen Audio- oder Videodatei wird die Schnittstelle der Transkriptionssteuerung verwendet, um die Transkription zu erhalten. Die Transkriptionsteuerung verwendet dafür zuerst die Schnittstelle des Format-Handlers, um das Material in ein für DeepSpeech korrektes Format zu konvertieren. Schließlich kann die von DeepSpeech bereitgestellte Schnittstelle verwendet werden, um den eigentlichen Text zu transkribieren.

Über die Editor-UI wird ebenfalls eine Schnittstelle des Plug-in-Managers verwendet. Über diese Schnittstelle sollen sich Plug-ins ausführen und verwalten lassen. Für die Ausführung soll der Aufruf durch den Plug-in-Manager an das jeweilige Plug-in weitergeleitet werden. Dafür soll der Plug-in-Manager eine Möglichkeit zur Verfügung stellen, über die sich Plug-ins bei diesem registrieren können.

5 Programmiersprache und Plattformwahl

Für die Umsetzung wurde die Programmiersprache Python gewählt. So hat sich in der ersten Recherche ergeben, dass viele mögliche Tools zur Verarbeitung des Textes eine Python-API bereitstellen. Dazu zählen zum Beispiel das `Natural Language Toolkit` [88] oder `spaCy` [2], mit welchen ein natürlichsprachlicher Text verarbeitet werden kann. Eine einfache Rechtschreibprüfung kann beispielsweise ebenso über Python mittels des `LanguageTool` [60] durchgeführt werden. Python wurde natürlich auch aus persönlicher Präferenz gewählt. So soll auf bestehende Erfahrung aufgebaut und diese durch das Projekt weiter ausgebaut und vertieft werden.

5.1 Wahl des GUI-Frameworks

Wie dem Python-Wiki [90] zu entnehmen ist, existiert eine Vielzahl von GUI-Frameworks für Python. Aus dieser Liste wurden die zuletzt Aktualisierten genauer betrachtet. Unter den Betrachteten stellt besonders das für ein Transkriptionsprogramm essenzielle Abspielen von Audio- und Videomaterial eine Herausforderung dar. So wird dafür häufig auf den `VLC`-Player oder den `mplayer` zurückgegriffen. Dies hat natürlich den Vorteil, dass eine große Vielfalt an Dateiformaten unterstützt werden kann. Problematisch ist jedoch, dass die Player für eine Verwendung auf dem System bereits installiert sein müssen. Auch wenn dies für den VLC-Player schon häufig der Fall ist, entstehen so jedoch weitere Abhängigkeiten. Hinzu kommt, dass die APIs der beiden Player nicht ganz ausgereift sind und deshalb beispielsweise Polling oder ein externer Timer eingesetzt wird, um die aktuell abgespielte Zeit anzuzeigen. [89], [22]

Um Abhängigkeiten so gering wie möglich zu halten, sollte das GUI-Framework also bereits einen Media-Player bereitstellen. Nach den ersten Tests ist dies unter den bisher ausgewählten lediglich bei PyQt5, PySide2 und wxPython der Fall. Während bei wxPython ebenfalls die aktuelle Zeit mittels Polling realisiert werden muss, bieten PyQt5 und PySide2 eine „schönere“ Implementation mittels Events (sogenannte `Signals`) an. [112], [33]

PyQt5 [15] und PySide2 [91] sind Python-Wrapper für das C++ Framework Qt [37], mit welchen sich plattformübergreifende Software entwickeln lässt. Das deutlich ältere PyQt5 wird von Riverbank Computing entwickelt und ist unter der GNU GPLv3 Lizenz sowie der Riverbank Commercial License verfügbar. Da Nokia, der damalige Besitzer von Qt, jedoch eine Python-API unter der LGPL-Lizenz haben wollte, entschied man sich eine eigene Implementierung unter dem Namen PySide zu veröffentlichen. Während PyQt anfangs deutlich fortgeschrittener war, sind die aktuellen Versionen PyQt5 und PySide2 kaum zu unterscheiden. Häufig reicht es, den Import in bestehenden Programmen für eine Portierung zu tauschen. Da die LGPL-Lizenz weniger streng und nach Qt für studentische oder akademische Zwecke empfohlen wird, wird

das Projekt mit PySide2 umgesetzt. [15], [91], [32], [36], [34]

5.2 Media-Player Beispiel

Folgend ein Beispiel-Media-Player, um die Funktionsweise von PySide2 kurz zu erläutern. In PySide2 (bzw. Qt) sind die Elemente der GUI sogenannte `QWidgets`. Diese QWidgets können mithilfe von Layouts mit anderen QWidgets befüllt werden. Für ein eigenes QWidget muss also, wie in Programmausdruck 3 zu sehen ist, zuerst die Klasse QWidget abgeleitet und schließlich die `init`-Methode überschrieben werden.

Danach (Z. 5-9) werden die einzelnen Buttons erzeugt. Mithilfe der `connect`-Methode werden übergebene Methoden bei Veränderung der jeweiligen Variable mittels Signale aufgerufen. Durch Zeile 9 wird zum Beispiel die Methode `on_play` (Z. 33-34) beim Drücken (Variable `clicked`) des Play-Buttons aufgerufen. Die erstellten Buttons werden daraufhin (Z. 12-13) in richtiger Reihenfolge einem horizontalen Layout hinzugefügt. Um das eigentliche Video und die Kontrollleiste übereinander anzuzeigen, wird zusätzlich ein vertikales Layout verwendet (Z. 16-19). Damit das Ergebnis bei Nutzung des eigenen QWidget angezeigt wird, muss danach das Layout für dieses gesetzt werden (Z. 22).

Schließlich (Z. 25-30) wird der eigentliche Media-Player erzeugt. Dafür wird das zuvor erstellte QVideoWidget als Video-Output festgelegt. Damit die Seekbar an die aktuell verstrichene Zeit angepasst werden kann, wird die Positionsveränderung des Media-Players mit der `on_position_change`-Methode verbunden (Z. 27-28). In dieser (Z. 37-38) wird dann der Wert ebenfalls für die Seekbar gesetzt, sodass die abgespielte Zeit hier ersichtlich wird. Zuletzt (Z. 29-30) wird das zu abspielende Video festgelegt.

Mittels der `show`-Methode (Z. 43) kann der Media-Player angezeigt werden. Durch die `QApplication` werden die offenen Fenster verwaltet. Das Ergebnis des aus Programmausdruck 3 entstehenden Media-Players ist in Abbildung 24 zu sehen.

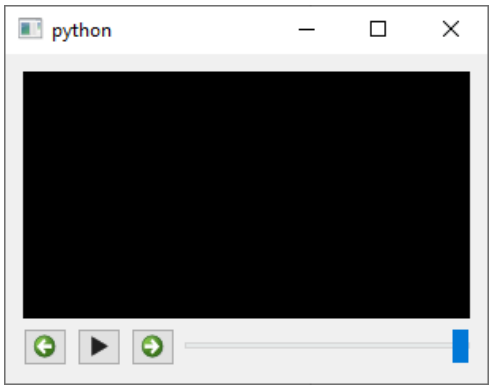


Abbildung 24: Einfacher Media-Player mittels PySide2

```
class MinimalMediaPlayer(QWidget):
    def __init__(self):
        super().__init__()

        # init Buttons
        self.play_btn = QPushButton()
        self.play_btn.setIcon(
            self.style().standardIcon(QStyle.SP_MediaPlay))
        self.play_btn.clicked.connect(self.on_play)

        # add Buttons and Seekbar to horizontal Layout
        self.h_box = QHBoxLayout()
        self.h_box.addWidget(self.play_btn)

        # add VideoWidget and horizontal Layout to vertical Layout
        self.video_widget = QVideoWidget()
        self.v_box = QVBoxLayout()
        self.v_box.addWidget(self.video_widget)
        self.v_box.addLayout(self.h_box)

        # set vertical layout
        self.setLayout(self.v_box)

        # init media player
        self.media_player = QMediaPlayer()
        self.media_player.setVideoOutput(self.video_widget)
        self.media_player.positionChanged.connect(
            self.on_position_change)
        self.media_player.setMedia(
            QMediaContent(QUrl.fromLocalFile(FILE_PATH)))

        # when play-button is clicked
        def on_play(self):
            self.media_player.play()

        # when media is played (position is changed)
        def on_position_change(self, position):
            self.seek_bar.setValue(position)

# running
app = QApplication(sys.argv)
window = MinimalMediaPlayer()
window.show()
app.exec_()
```

Programmausdruck 3: Media-Player umgesetzt mittels PySide2 (gekürzt)

6 Ergebnisse

Im Folgenden soll der entwickelte Editor (getauft „LazyTranscript“) vorgestellt werden. Dazu werden die einzelnen Masken und ihre Besonderheiten gezeigt und genauer beleuchtet.

6.1 Ein Projekt öffnen oder erstellen

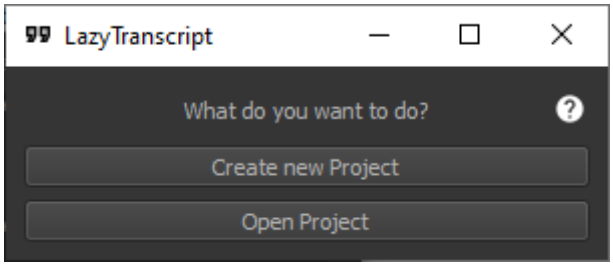


Abbildung 25: Start-Maske

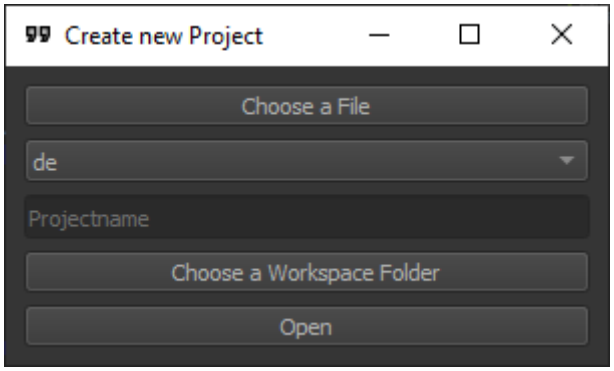


Abbildung 26: Ein-Projekt-erstellen-Maske

Sobald das Programm gestartet wurde, erscheint die in Abbildung 25 zu sehende Maske. Hier kann der Nutzer auswählen, ob er ein neues Projekt erstellen oder ein bestehendes öffnen möchte. Mithilfe des „?“-Buttons kann eine kleine Anleitung (Anhang A.5) geöffnet werden. Beim Öffnen eines bereits bestehenden Projektes muss der Nutzer über einen Dialog den jeweiligen Projektordner auswählen. Möchte der Nutzer ein neues Projekt erstellen, so wird die in Abbildung 26 sichtbare Maske geöffnet.

Um ein neues Projekt zu erstellen, muss zuerst das zu transkribierende Material ausgewählt werden. Danach muss der Nutzer aus den vorher selbst installierten Sprachen die Projektsprache bestimmen. Für die Installation einer Sprache muss lediglich im `models`-Verzeichnis ein Verzeichnis für die jeweilige Sprache angelegt und das Modell sowie ein passender Scorer in dieses verschoben werden (*V07*). Nach der Sprachauswahl müssen nachfolgend nur noch ein Projektname und der Zielspeicherort (`Workspace`) definiert werden.

Beim Betätigen des Open-Buttons wird zuerst der Projektordner im Zielspeicherort erstellt und dort eine Kopie des Ausgangsmaterials gespeichert. Dies soll verhindern, dass das Ausgangsmaterial später gelöscht, verschoben oder überschrieben wird. Anschließend wird es mittels `MoviePy` [114] in ein für DeepSpeech korrektes Format konvertiert. MoviePy verwendet dafür das beliebte Open Source Tool `FFmpeg` [31], wodurch eine Vielzahl von Audio- und Videoformaten unterstützt wird. Aus der Konvertierung resultiert eine wav-Datei mit einem einzigen Audio Channel, einer Samplingrate von 16 kHz und dem Audiocode `pcm_s16le`, wobei die Werte den DeepSpeech Beispielen [71] entnommen wurden.

Die Transkription dieser konvertierten Datei wird wie folgt durchgeführt. Zuerst wird das Audiomaterial in 30ms kleine Abschnitte zerteilt und schließlich mithilfe des WebRTC VAD (Kapitel 3.4) überprüft, in welchen dieser Abschnitte gesprochen wurde. Diese werden dann an DeepSpeech übergeben und transkribiert. Die Teiltranskriptionen werden daraufhin zu einer Gesamttranskription zusammengefasst und zuletzt in einer einfachen Textdatei gespeichert (*V10, V11*). Realisiert wurde der Transkriptionsprozess durch die Integration des `vad_transcriber`-Beispiels aus den DeepSpeech-Examples-Repository [72].

6.2 Eine Transkription bearbeiten

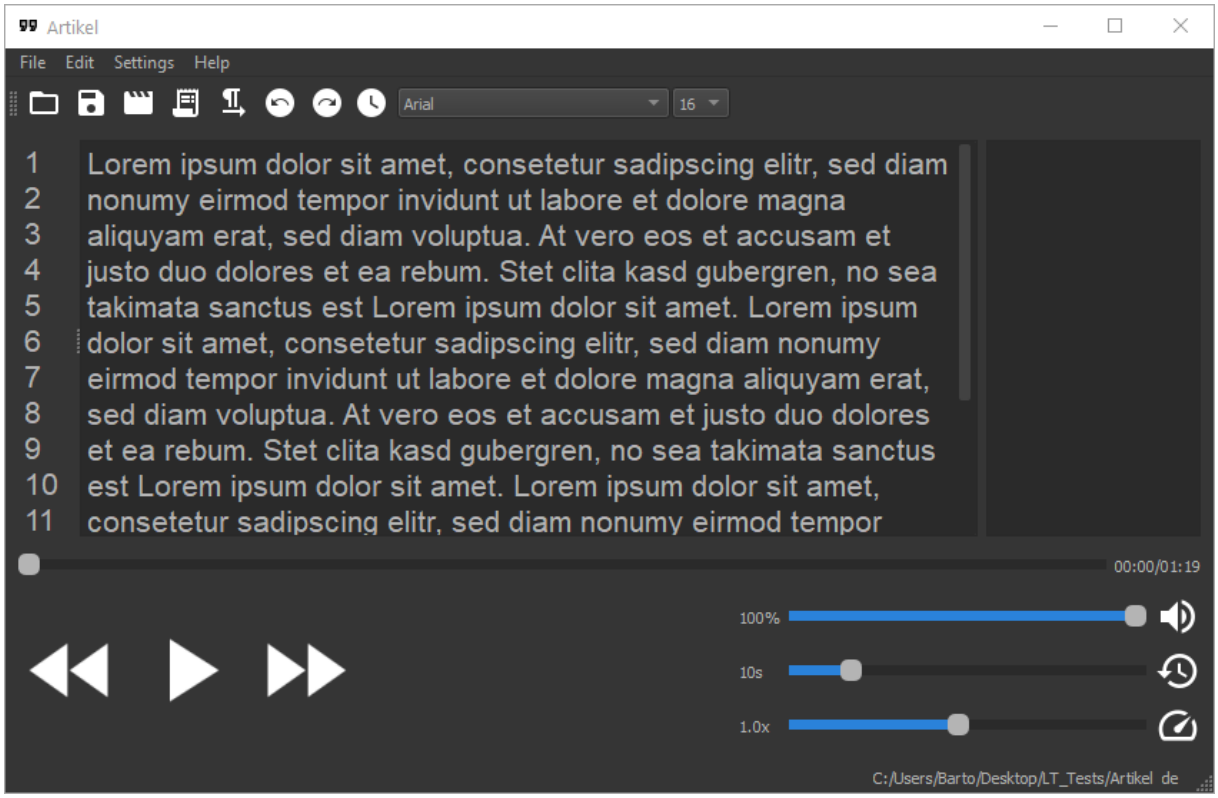


Abbildung 27: Editor-Maske

Nachdem die Transkription erstellt oder das Projekt geöffnet wurde, erscheint die eigentliche Editor-Maske (Abbildung 27, Anhang Abbildung 62). In der Menüleiste (Abbildung 28) findet der Nutzer neben klassischen Operationen wie Speichern, Öffnen, Kopieren und Einfügen (*V00, V01, V02*) auch die Einstellungen und die Hilfe (Anhang A.5). Einige dieser Funktionen lassen sich ebenfalls über die darunterliegende Toolbar (Abbildung 29) oder mit einer Tastenkombination ausführen. Zusätzlich kann der Nutzer über die Toolbar beim Transkribieren eines Videos das Videobild öffnen, die Textbausteine aktivieren, Leerzeichen und Umbrüche anzeigen, die Transkription Wort für Wort durchlaufen, einen Zeitstempel einfügen (*V13*) oder die Schriftart

und -größe (*V04*) nach seinen Wünschen anpassen.

Darunter befindet sich das Textfeld mit der eigentlichen Transkription und der Zeilennummerierung (*V03*). Diese kann bei Bedarf ein- und ausgeblendet werden. Im unteren Bereich dieser Maske (Abbildung 30) befindet sich die Mediensteuerung (*V05*). Neben einer klassischen Seekbar und einem Play- und Pause-Button, kann der Nutzer hier auch vor- und zurückspulen. Diese Funktionen lassen sich ebenfalls mittels verschiedener Tastenkombinationen ausführen. Auf der rechten Seite kann der Nutzer die Lautstärke, das Spulintervall sowie die Geschwindigkeit (*V06*) nach seinen Wünschen anpassen.

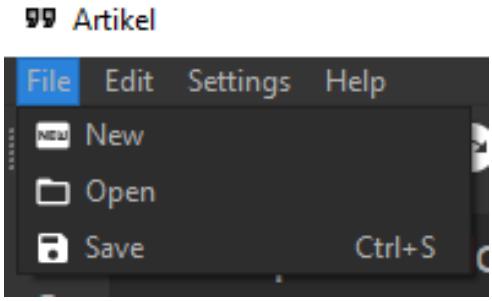


Abbildung 28: Menüleiste: Beispiel der klassischen Operationen

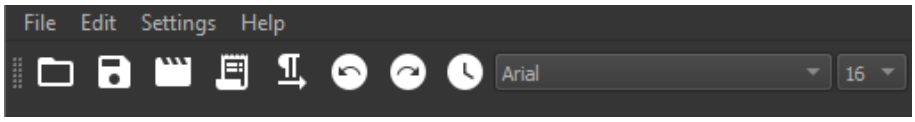


Abbildung 29: Menüleiste und Toolbar

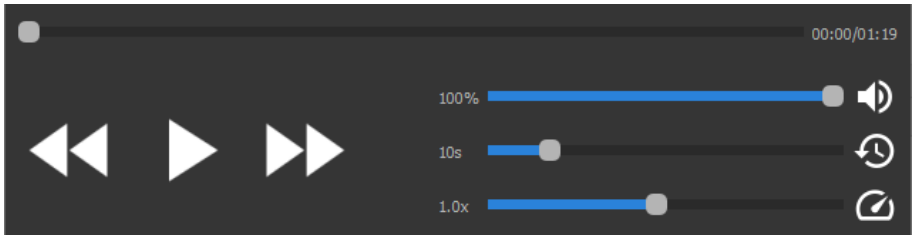


Abbildung 30: Editor-Maske: Mediensteuerung

6.3 Textbausteine

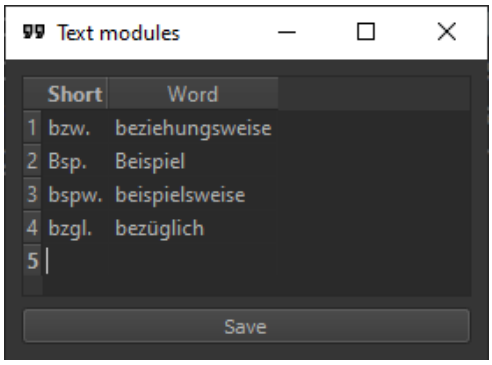


Abbildung 31: Textbaustein-Maske

Um neue Textbausteine hinzuzufügen, muss über die Menüleiste unter den Einstellungen die Textbaustein-Maske (Abbildung 31) aufgerufen werden. Der Nutzer kann hier über eine einfache Tabelle neue Textbausteine definieren (*V09*). Damit diese nicht mehrfach für unterschiedliche Projekte angelegt werden müssen, gelten diese projektübergreifend. Über die Toolbar können die Textbausteine beliebig ein- oder ausgeschaltet werden. Sind sie eingeschaltet, so werden sie automatisch durch das definierte Wort ersetzt.

6.4 Wort-für-Wort-Bearbeitung

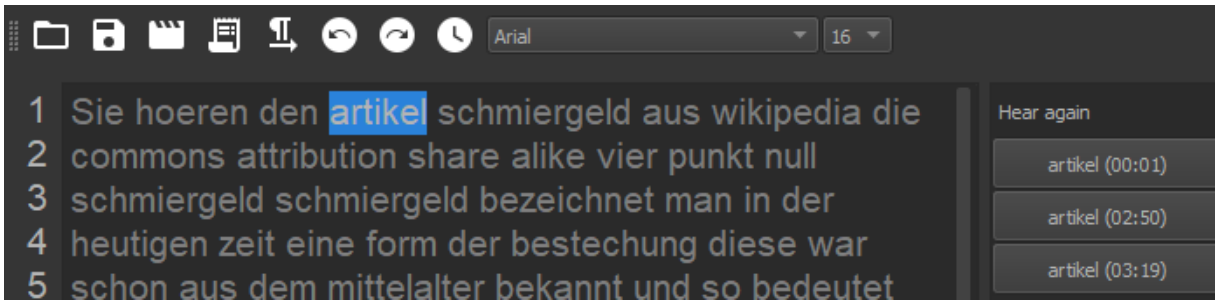


Abbildung 32: Wort-für-Wort-Bearbeitung

Um eine Transkription Wort für Wort zu durchlaufen, müssen entweder die Pfeil-Buttons links neben dem Uhr-Button oder die F5- und F6-Tasten gedrückt werden. Wie in Abbildung 32 zu sehen ist, wird das aktuelle Wort markiert und im rechten Bereich der Maske Vorschläge zur Ver- und Bearbeitung angezeigt. Falls das Wort in der Ursprungstranskription existiert, wird hier standardmäßig die Möglichkeit angeboten, die verschiedenen Stellen, an denen es vorkommt, noch einmal anzuhören (*N05*). Mögliche weitere Korrekturen durch die Plug-ins werden in Kapitel 8 vorgestellt und genauer erläutert.

Im Gegensatz zum Wireframe in Abbildung 20 fiel die Entscheidung gegen ein Kontextmenü, um den Text nicht zu verdecken und Mausbewegungen zu verringern. Je nach gewähltem Button wird beim erneuten Drücken das vorherige (F5) oder das nächste (F6) Wort markiert. Mit der Escape-Taste kann die Wort-für-Wort-Bearbeitung beendet werden.

6.5 Einstellungen und Lizenzen

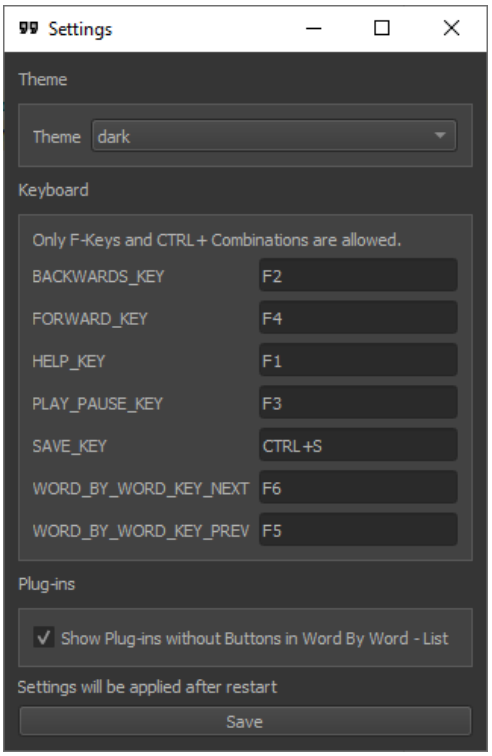


Abbildung 33: Einstellungs-Maske

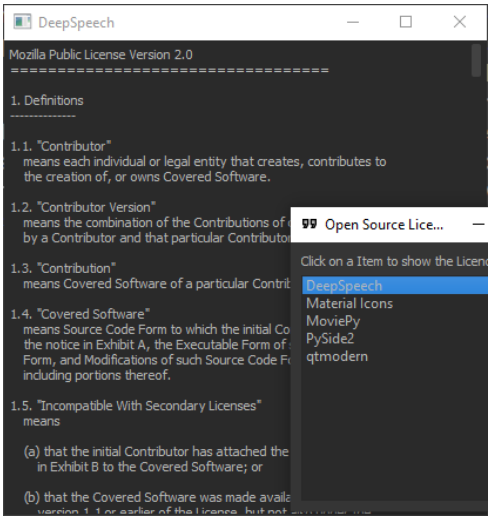


Abbildung 34: Lizenz-Maske

Wie zu sehen ist, kann in den Einstellungen (Abbildung 33) zwischen einem hellen und einem dunklen Design, bereitgestellt von `qtmodern` [61], gewählt werden. Außerdem ist es hier möglich, die Tastenkombinationen anzupassen (*V08*). Dabei werden ebenso die Tastenkombinationen der Plug-ins angezeigt. Da es möglich ist, dass einige Plug-ins nur eine Nachricht oder Überschrift anzeigen und keine Buttons zurückgeben, können diese der Übersicht halber ausgeblendet werden.

Um die genutzten Open Source Bibliotheken einzusehen, kann die Lizenz-Maske (Abbildung 34) aufgerufen werden. Mit einem Klick auf einen Eintrag öffnet sich die hinterlegte Open Source Lizenz.

7 Schwächen und Fehler in der Transkription

Im Folgenden soll eine Transkription genauer untersucht werden, um mögliche Schwächen und Fehler zu finden. Um diese Untersuchung möglichst unbefangen und nicht zum Vor- oder Nachteil für DeepSpeech durchzuführen, wurde aus der Liste der gesprochenen Wikipedia-Artikel [108] der Artikel „Schmiergeld“ [109], gesprochen von Matthias Senke (Benutzername KnubbellTTV), ausgewählt. Für die deutsche Sprache wird das DeepSpeech-Modell aus dem DeepSpeech-Polyglot Projekt des Jaco-Assistant unter der GNU LGPLv3 Lizenz verwendet [53]. Die Audioaufnahme ist unter der Creative Commons Attribution-Share Alike 4.0 [14] und der Artikel unter der Creative Commons Attribution-Share Alike 3.0 Unported [13] lizenziert.

Der erste Ausschnitt aus dem Artikel lautet ohne Überschriften wie folgt:

„[...] *Als Schmiergeld bezeichnet man in der heutigen Zeit eine Form der Bestechung. Diese war schon aus dem Mittelalter bekannt und so bedeutet schmieren auch bestechen (jemand die Hand schmieren). Das Wort Smeergeld im Sinn von Bestechung ist um 1700 im Niederdeutschen bezeugt. Begrifflich kann Schmiergeld auch in der Form jedes sonstigen materiellen Vorteils verstanden werden.*

Bis zum Steuerjahr 1995 konnten in Deutschland Schmiergelder als Betriebsausgaben von den zu versteuernden Einnahmen abgezogen werden. Seither fallen Schmiergelder unter die Betriebsausgaben, die den Gewinn nicht mindern dürfen. § 4 Abs. 5 Nr. 10 Einkommensteuergesetz besagt hierzu, dass unter anderem die folgenden Betriebsausgaben den Gewinn nicht mindern dürfen [...]“ [109]

Die von DeepSpeech (mit VAD) resultierende Transkription dieses Abschnittes, ebenfalls ohne Überschriften und Hinweise des Sprechers, lautet wie folgt:

„[...] ____ *schmiergeld bezeichnet man in der heutigen zeit eine form der bestechung_diese war schon aus dem mittelalter bekannt und so bedeutet schmieren auch bestechen _jemand die handspiel__das wort mehr geld im sinn von bestechung ist um siebzehn hundert im niederdeutschen bezeugt_begrifflich kann schmiergeld auch in der form jedes sonstigen materiellen vorteils verstanden werden_*

bis zum steuerjahr neunzehn hundert fuenfundneunzig konnten in deutschland schmiergelder als betriebsausgaben von dem zu versteuernden einnahmen abgezogen werden_seither fallen schmiergelder unter die betriebsausgaben_die dinge wenn nicht mindern duerfen paragraph absender meister gesetz besagt hierzu_das unter anderem die folgenden betriebsausgaben den gewinn nicht mindern duerfen“ (ebenfalls unter [14] lizenziert)

Die gefundenen Fehler und Schwächen sind jeweils nach der in Tabelle 7 zu sehenden Farbcodierung markiert. Die fehlende Großschreibung hat zur besseren Übersichtlichkeit keine eigene Farbe.

Fehlendes oder falsches Wort	Fehlender Punkt am Satzende
Fehlende Klammer	Ausgeschriebene Zahl
Ausgeschriebener Umlaut	Fehlendes Komma

Tabelle 7: Farbcodierung / Legende für den von DeepSpeech transkribierten Text

Der auffälligste Fehler im Text ist die fehlende Großschreibung. Diese sollte sich mit einer Rechtschreibprüfung, wie dem bereits genannte LanguageTool [60] oder mit einer Wortarterkennung, wie sie von dem Natural Language Toolkit [88] oder dem HanoverTagger [105] bereitgestellt wird, korrigieren lassen.

Um fehlende oder falsche Wörter zu korrigieren, könnte versucht werden, einen Alternativvorschlag einer anderen Speech-To-Text-Engine breitzustellen. Außerdem könnte dem Nutzer die Möglichkeit gegeben werden, Worte einfach zu ersetzen oder anzuhängen.

Während ausgeschriebene Umlaute leicht zu ersetzen sind, ist dieser Prozess bei ausgeschriebenen Zahlen etwas komplizierter. Erschwert wird dies besonders durch den unterschiedlichen Aufbau der Zahlen in den verschiedenen Sprachen („einundzwanzig“ vs. „twentyone“). Hier muss später erörtert werden, ob einer der verschiedenen Wort-zu-Zahl-Konverter verwendet, portiert oder selbst implementiert werden kann.

Fehlende Klammern, Punkte, Kommata und andere Sonderzeichen wie Ausrufe- und Fragezeichen sollten ebenfalls leicht vom Nutzer an das jeweilige Wort angehängt werden können. Gegebenenfalls lässt sich hier auch eine automatische Lösung finden.

Insgesamt zeigt sich jedoch, dass die erste Transkription von DeepSpeech ein grundsätzlich gutes Ergebnis liefert. So beträgt die WER 15.03% und liegt dabei knapp unter der von DeepSpeech-Polyglot angegebenen WER von 16.01870% [53]. Die jeweiligen Fehlerarten (Vergleiche Tabelle 4) lassen sich im Anhang in Tabelle 9 einsehen und wurden mithilfe des `WERinPython`-Projektes [113] erstellt. Da die Transkription im Vordergrund steht, sollen die genannten Punkte vor den (anderen) nicht verpflichtenden Anforderungen (Tabelle 6) als Plug-ins umgesetzt werden.

8 Plug-ins

Im Folgenden werden die Plug-ins genauer erläutert. Dafür werden zuerst die Plugin-in-Struktur und schließlich die im Laufe der Masterarbeit entstandenen Plug-ins beschrieben (*V12*).

8.1 Plug-in-Struktur

Anhand der bisherigen Kapitel bestehen an die Plug-in-Struktur folgende zwei Anforderungen. Als Erstes müssen die Pug-ins die Möglichkeit haben, verschiedene Korrekturvorschläge für die Wort-für-Wort-Bearbeitung (Abbildung 32) bereitzustellen. Ein Beispiel hierfür sind mögliche Rechtschreibkorrekturen, wie sie in Abbildung 35 zu sehen sind. Als Zweites muss es den Plug-ins möglich sein, den kompletten Text zu be- und verarbeiten. Ein Beispiel hierfür wäre das Ersetzen aller ausgeschriebenen Umlaute, um eines der in Kapitel 7 aufgedeckten Probleme zu beheben. Die Korrektur sollte hier über ein Button in der Toolbar, wie es in den Wireframes (Kapitel 4.5) definiert wurde, ausgelöst werden. Ein Beispiel verschiedener Toolbar-Plug-ins ist in Abbildung 36 zu sehen.

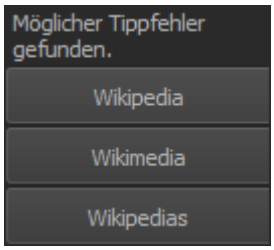


Abbildung 35: Beispiel: Rechtschreibkorrektur in der Wort-für-Wort-Bearbeitung

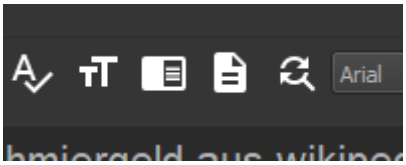


Abbildung 36: Beispiel: Toolbar-Plug-ins

Um dies zu realisieren wurde für die Plug-in-Struktur, wie in Kapitel 4.6 beschrieben, ein Plug-in-Manager und ein Plug-in-Interface implementiert. Nachfolgend werden diese Komponenten und die daraus resultierende Struktur genauer erläutert und vorgestellt.

8.1.1 Plug-in Interface

```
1  class PluginSignal(QObject):
2      create_button = Signal(str, int, dict)
3      execute_action = Signal(dict)
4
5  class IPlugin():
6      def __init__(self, plugin_manager):
7          self.plugin_manager = plugin_manager
8          self.sig = PluginSignal()
9
10     def project_loaded(self):
11         pass
12
13     def get_word_action(self, word: str, word_meta_data: List[dict],
14                         word_pos: int):
15         pass
16
17     def get_toolbar_action(self, parent) -> List[QAction]:
18         return []
19
20     def get_name(self) -> str:
21         return None
```

Programmausdruck 4: Plug-in-Interface

Damit ein Plug-in in LazyTranscript eingebunden werden kann, muss es das in Programmausdruck 4 zu sehende Interface (in Python eine Klasse) implementieren. Die Methoden dieses Interfaces haben dabei die folgende Bedeutung und Verwendung:

In der `init`-Methode (Z. 6) wird das jeweilige Plug-in initialisiert. Hierfür muss der Plug-in-Manager übergeben und ein sogenanntes `PluginSignal` erzeugt werden. Der Plug-in-Manager kann später verwendet werden, um den Text und bestimmte Wörter zu verarbeiten oder andere Informationen abzufragen (siehe auch Kapitel 8.1.2).

Wie der Name vermuten lässt, wird die `project_loaded`-Methode (Z. 10) aufgerufen, sobald ein Projekt geöffnet wurde. Diese Methode kann also verwendet werden, um bestimmte projekt- oder sprachspezifische Einstellungen für das Plug-in durchzuführen.

Die `get_word_action`-Methode (Z. 13) wird für jedes Wort bei der Wort-für-Wort-Bearbeitung (Kapitel 6.4) aufgerufen. Das jeweilige Plug-in erhält hier das aktuell markierte Wort, seine textuelle Position und (falls vorhanden) die Meta-Daten. Die Meta-Daten enthalten dabei die Start- und Endzeiten des Wortes im Audio- bzw. Videomaterial. Da die genaue Position im Ausgangsmaterial nicht festgestellt werden kann, werden die Meta-Daten aller Vorkommnisse des Wortes übergeben.

Aufgabe des Plug-ins ist es nun `QPushButtons` zu erzeugen und an den Plug-in-Manager zurückzugeben (vergleiche Abbildung 37). QPushButtons sind Buttons im QT-Framework, welche beim Drücken eine vorher definierte Methode ausführen. In dieser verknüpften Methode findet die eigentliche Korrektur mithilfe verschiedener Methoden (Tabelle 8) des Plug-in-Managers statt. Die erzeugten Buttons werden schließlich als Korrekturvorschläge (Abbildung 32) angezeigt.

Da die Korrektur und die Erzeugung der Korrekturvorschläge sehr zeitintensiv sein kann, stellt das Plug-in-Interface mithilfe des PluginSignals (Z. 8) eine Schnittstelle zur Verfügung, über die der Hauptthread benachrichtigt werden kann. Beim Aufruf der `get_word_action`-Methode kann beispielsweise ein `QThread` gestartet werden, welcher die möglichen Korrekturvorschläge im Hintergrund erzeugt. Die Ergebnisse können dann mithilfe des `create_button`-Signals (Z. 2) an eine Methode des Plug-ins weitergeben werden. Dort kann dann im Hauptthread die eigentliche Erzeugung der QPushButtons stattfinden.

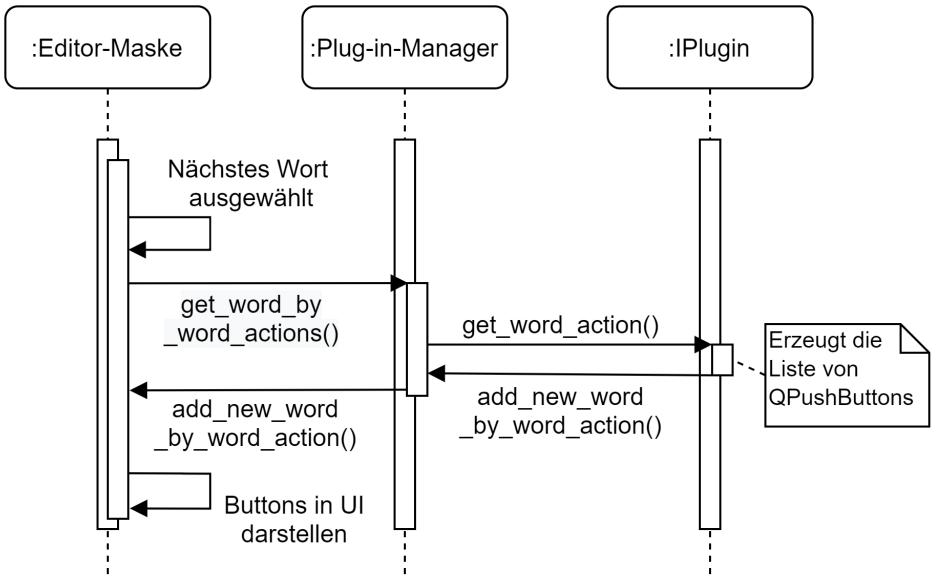


Abbildung 37: Sequenzdiagramm zur Visualisierung der get_word_action-Methode

Die `get_toolbar_action`-Methode (Z. 17) ist der `get_word_action`-Methode in weiten Teilen sehr ähnlich. Statt QPushButtons werden hier allerdings `QActions` erzeugt und zurückgegeben (vergleiche Abbildung 38). Bei diesen QActions handelt es sich um die Buttons, die in der Toolbar angezeigt werden (Abbildung 29). Wie ein QPushButton, kann eine QAction mit einer Methode verknüpft werden. Über diese Methode kann beispielsweise ein neues Fenster geöffnet oder eine Korrektur des ganzen Textes durchgeführt werden.

Ähnlich wie das `create_button`-Signal kann auch das `execute_action`-Signal (Z. 3) beim Drücken eines Toolbar-Buttons verwendet werden. Hier wird durch das Drücken des Toolbar-

Buttons ein QThread gestartet und die Korrektur im Hintergrund durchgeführt. Sobald diese abgeschlossen ist, wird der neue Text mithilfe des Signals an eine Methode im Hauptthread zurückgegeben. Diese Methode wendet daraufhin die Korrektur mithilfe des Plug-in-Managers an.

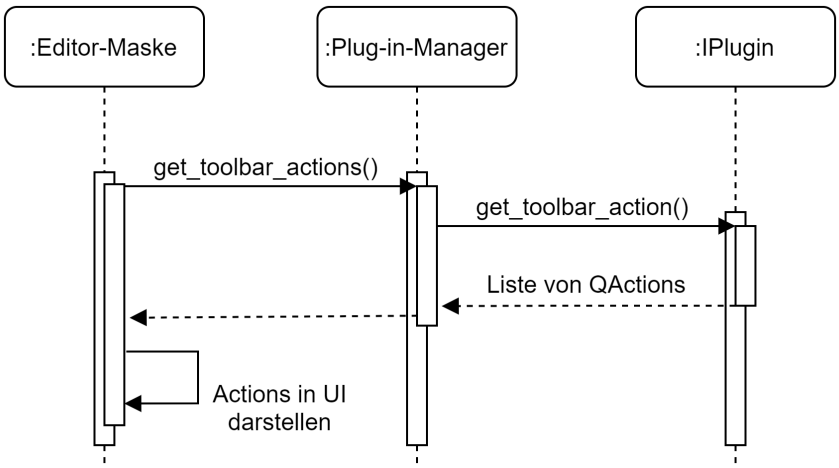


Abbildung 38: Sequenzdiagramm zur Visualisierung der get_toolbar_action-Methode

Über die `get_name`-Methode wird der Plug-in-Name definiert. Er kann als Überschrift für die Buttons in der Wort-für-Wort-Bearbeitung verwendet werden oder dient zur Persistierung der Plug-in-Einstellungen.

8.1.2 Plug-in-Manager

Wie bereits in Kapitel 4.6 erläutert, ist der Plug-in-Manager hauptsächlich für die Initialisierung der Plug-ins und die Weiterleitung von Aufrufen zuständig. Damit die Plug-ins durch den Plug-in-Manager initialisiert werden können, müssen sie eine bestimmte Struktur aufweisen. Natürlich muss das Plug-in das bereits erläuterte Plug-in-Interface implementieren. Zusätzlich dazu muss die Klasse den Namen `Plugin` haben und in einer Python-Datei mit der Endung „_lt_plugin“ enthalten sein.

Damit die einzelnen Dateien der Plug-ins gebündelt sind, hat jedes Plug-in sein eigenes Unterverzeichnis, in welchem beliebig viele Dateien oder Verzeichnisse abgelegt werden können. Falls das Plug-in auf Open Source Bibliotheken zurückgreift, kann es die verschiedenen Lizenzen im `licences`-Verzeichnis ablegen. Diese werden dann automatisch durch den Plug-in-Manager in der Lizenz-Maske (Abbildung 34) dargestellt. Die Initialisierung der Plug-ins ist so realisiert, dass durch einfaches Hinzufügen oder Entfernen des Plug-in-Verzeichnisses, ein Plug-in installiert oder deinstalliert werden kann. Die komplette Verzeichnis-Struktur der Plug-ins ist in Abbildung 39 noch einmal dargestellt.

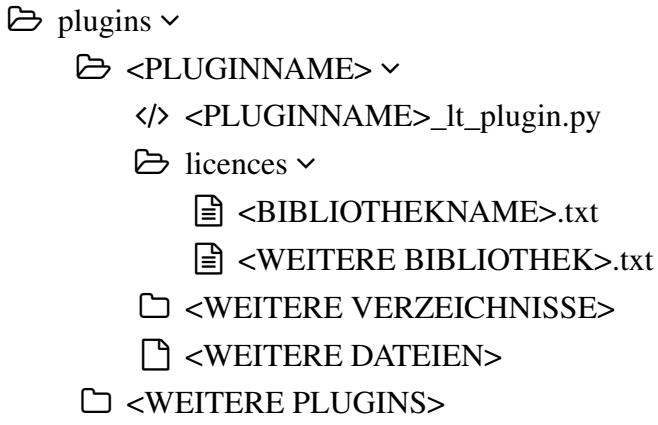


Abbildung 39: Plug-in-Verzeichnis-Struktur

Nachdem das Programm gestartet wurde, beginnt der Plug-in-Manager das plugin-Verzeichnis zu durchlaufen und die einzelnen Plug-ins mittels der Python-Standardbibliothek `importlib` [35] zu importieren. Dafür wird für jede gefundene „_lt_plugin“-Datei der in Programmausdruck 5 zu sehende Ablauf durchgeführt.

```
1  import importlib.util as ilu
2  try:
3      temp_spec = ilu.spec_from_file_location(modulename, module_path)
4      temp_module = ilu.module_from_spec(temp_spec)
5      temp_spec.loader.exec_module(temp_module)
6      temp_plugin = temp_module.Plugin(self)
7      if issubclass(type(temp_plugin), IPlugin):
8          self.plugin_list.append(temp_plugin)
9          print("Imported {} as a Plugin".format(file))
10 except BaseException as e:
11     print("Error while importing {} with Exception {}".format(file, e))
```

Programmausdruck 5: Import einer Python-Datei als Modul

Zuerst wird für die „_lt_plugin“-Datei ein `ModuleSpec` erzeugt (Z. 3). Das ModuleSpec enthält alle wichtigen Informationen, welche für das Laden und Importieren eines Moduls (einer Python-Datei) benötigt werden. Mithilfe des ModuleSpecs wird daraufhin das eigentliche Modul erst erzeugt (Z. 4) und anschließend ausgeführt (Z. 5). Danach können die Objekte und Funktionen des Moduls verwendet werden. In diesem Fall wird zuerst eine Plug-in-Instanz erzeugt (Z. 6) und danach überprüft, ob es sich dabei um eine Subklasse des Plug-in-Interfaces handelt (Z. 7). Sollte dies zutreffen, so wird das Plug-in der Liste von Plug-ins angehängt (Z. 8). [35]

Sobald ein Projekt geöffnet wurde, kann die Editor-Maske (Abbildung 27) über den Plug-in-Manager die Toolbar-Buttons und die Korrekturvorschläge der einzelnen Plug-ins abfragen. Dafür ruft der Plug-in-Manager für alle initialisierten Plug-ins die jeweiligen Methoden (Kapitel 8.1.1) auf. Wie in Kapitel 4.6 und 8.1.1 beschrieben, dient der Plug-in-Manager für die Plug-ins als Schnittstelle zum eigentlichen Text im Editor. Für die Korrektur und Auswertung des Textes stellt der Plug-in-Manager deshalb die in Tabelle 8 stehenden Methoden zur Verfügung. Die Aufrufe werden dabei an den Editor weitergeleitet und müssen dementsprechend dort ebenfalls implementiert sein.

Neben den Lizenzen der Plug-ins verwaltet der Plug-in-Manager aber auch die Tastenkombinationen der Toolbar-Buttons. So verhindert er Duplikate mit den bereits festgelegten und fügt diese den Einstellungen hinzu. Über die Einstellungs-Maske (Abbildung 33) können dadurch auch für Plug-ins eigene Tastenkombinationen definiert werden.

Methodenname	**Funktion**
get_selection	Gibt den aktuell markierten Text zurück.
replace_selection	Ersetzt den aktuell markierten Text mit dem übergebenen.
get_text	Gibt den kompletten Text zurück.
set_text	Ersetzt den kompletten Text durch den übergebenen.
set_text_with_line _breaks	Ersetzt den kompletten Text durch den übergebenen und versucht dabei die Zeilenumbrüche beizubehalten.
get_word_at	Gibt das Wort an der übergebenen Position zurück.
set_word_at	Setzt oder ersetzt das übergebene Wort an der übergebenen Position.
set_hint_text	Zeigt den übergebenen Text im unteren linken Bereich des Editors an.
get_setting	Gibt die abgefragte Einstellung zurück.
get_language	Gibt die Projektsprache zurück.
get_project_folder_path	Gibt den aktuellen Projektpfad zurück.

Tabelle 8: Weitere Methoden des Plug-in-Managers

8.1.3 Beispiel Plug-ins

Nachfolgend soll diese Struktur anhand richtiger Beispiele in Form des `word_to_number`- sowie des `umlauts`-Plug-ins gezeigt werden.

Das word_to_number-Plug-in soll bei der Wort-für-Wort-Bearbeitung einen Button zur Verfügung stellen, welcher die ausgeschriebenen Zahlen in ihr numerisches Äquivalent konvertiert. Wie in Programmausdruck 6 zu sehen ist, wird dafür zuerst in der `init`-Methode (Z. 3) die `super`-Methode aufgerufen und der Konverter erzeugt.

```
1  class Plugin(IPlugin):
2
3      def __init__(self, plugin_manager):
4          super(Plugin, self).__init__(plugin_manager)
5          self.conv = WordToNumberConverter()
6
7      def get_word_action(self, word, word_meta_data, word_pos):
8          if "de" in self.plugin_manager.get_language().lower():
9              number = self.conv.map_word_to_number(word.lower())
10
11         if number == word.lower():
12             return
13
14         btn = QPushButton(str(number))
15         btn.clicked.connect(lambda : self.plugin_manager.set_word_at(
16                                         number, word_pos, True))
17
18         self.plugin_manager.add_new_word_by_word_action(
19                         [btn], self.get_name(), word, word_pos)
20
21     def get_name(self) -> str:
22         return "Numbers"
```

Programmausdruck 6: word_to_number-Plug-in (gekürzt)

Beim Aufruf der `get_word_action`-Methode (Z. 7 ff.) wird zuerst die Sprache überprüft und schließlich versucht, das Wort zu konvertieren. Handelt es sich nicht um eine Zahl und dies schlägt fehl (Z. 11), wird die Methode beendet. War die Konvertierung jedoch erfolgreich, wird ein QPushButton mit der Zahl erzeugt (Z. 14) und mittels der `connect`-Methode mit der Korrektur verknüpft (Z. 15). In diesem Fall wird dafür mittels des Schlüsselwortes `lambda` eine anonyme Funktion erstellt, welche die `set_word_at`-Methode des Plug-in-Managers aufruft und somit das Wort durch die Zahl ersetzt. Zuletzt (Z. 18) wird dieser Button an den Plug-in-Manager zurückgegeben.

Das umlauts-Plug-in soll in der Toolbar einen Button anzeigen, welcher die ausgeschrieben Umlaute konvertiert. Wie auch im vorherigen Beispiel wird hier (Programmausdruck 7) zuerst in der `init`-Methode (Z. 3) die super-Methode aufgerufen und das Set der Umlaute erzeugt.

```
1  class Plugin(IPlugin):
2
3      def __init__(self, plugin_manager):
4          super(Plugin, self).__init__(plugin_manager)
5          self.chars = {"ae" : "ä", "oe" : "ö", "ue" : "ü",
6                        "Ae" : "Ä", "Oe" : "Ö", "Ue" : "Ü"}
7
8      def get_toolbar_action(self, parent_window) -> List[QAction]:
9          icon_path = os.path.join(c.PLUGIN_PATH, "umlauts", "icons",
10                                  self.plugin_manager.theme, "search.png")
11         action = QAction(QIcon(icon_path),
12                          "Replace umlauts", parent_window)
13         action.triggered.connect(self.replace_umlauts)
14         return [action]
15
16     def replace_umlauts(self):
17         text = self.plugin_manager.get_text()
18         for char in self.chars:
19             text = text.replace(char, self.chars[char])
20         self.plugin_manager.set_text(text)
21
22     def get_name(self) -> str:
23         return "Replace Umlauts"
```

Programmausdruck 7: umlauts-Plug-in

In der `get_toolbar_action`-Methode (Z. 8) wird dann die QAction erstellt. Hierfür wird erst das passende Icon aus dem umlauts-Verzeichnis herausgesucht (Z. 9) und nach Erzeugung der QAction (Z. 11) die `replace_umlauts`-Methode mit dieser verknüpft. Diese Methode führt beim Drücken des Buttons die eigentliche Korrektur des Textes durch. Danach (Z. 14) wird die QAction dem Plug-in-Manager zurückgegeben.

Beim Durchführen der Korrektur (Z. 16) holt sich das Plug-in den aktuellen Text mithilfe des Plug-in-Managers (Z. 17). Danach werden alle ausgeschriebenen Umlaute aus dem anfangs definierten Set im Text ausgetauscht. Zuletzt wird der alte Text mittels der `set_text`-Methode des Plug-in-Managers mit der Korrektur ersetzt (Z. 20).

8.2 Plug-ins zur Verbesserung des Ergebnisses

Folgend werden die Plug-ins vorgestellt, welche die ursprüngliche Transkription von DeepSpeech (siehe Kapitel 7 und besonders Tabelle 7) verbessern. Zum Teil werden durch diese auch schon einige der nicht verpflichtenden Anforderungen (Tabelle 6) umgesetzt.

8.2.1 Wort

Mithilfe des Wort-Plug-ins können Wörter bei der Wort-für-Wort-Bearbeitung vorangestellt, angehangen, großgeschrieben, ersetzt, verkettet oder entfernt werden. Bei Bedarf wird dafür ein Textdialog (Abbildung 41) angezeigt, in welchen der Nutzer das Wort, welches eingefügt werden soll, eingeben muss.

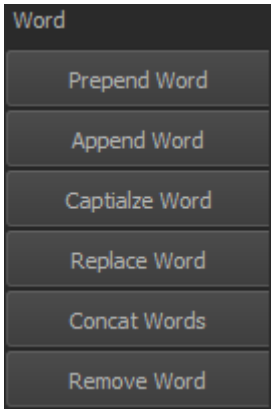


Abbildung 40: Wort-Plug-in

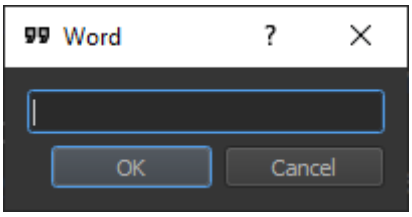


Abbildung 41: Wort-Plug-in Dialog

8.2.2 Wort-zu-Zahl-Konvertierung

Da DeepSpeech Zahlen nur ausgeschrieben ausgibt, dient dieses Plug-in dazu, diese in numerische Werte zu konvertieren. Für die deutsche Sprache wurde die „Word to Number“ [49] Bibliothek von IBM verwendet. Es handelt sich hierbei um eine JavaScript Bibliothek, welche für das Plug-in zu Python portiert wurde. Für die englische Sprache wurde hingegen die „Word to Number“ [77] Bibliothek von Akshay Nagpal eingesetzt. Da nicht jede Zahl konvertiert werden soll, handelt es sich bei diesem Plug-in ebenfalls um ein Plug-in für die Wort-für-Wort-Bearbeitung.

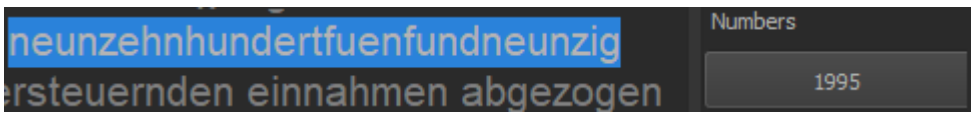


Abbildung 42: Wort-zu-Zahl-Plug-in

8.2.3 Satzzeichen

Dieses Plug-in ermöglicht es dem Nutzer die Satzzeichen . , ! ? ( ) an das aktuell ausgewählte Wort bei der Wort-für-Wort-Bearbeitung anzuhängen und realisiert damit die nicht verpflichtende Anforderung *N02*. Je nach ausgewählten Zeichen wird das Folgewort automatisch großgeschrieben. Mit diesem Plug-in soll es möglich sein, die in der Transkription fehlenden Satzzeichen wiederherzustellen.

An dieser Stelle wurde auch überlegt, eine automatische Lösung zu integrieren. Diese wurde aber aus verschiedenen Gründen verworfen. Einerseits bieten die gefundenen automatischen Lösungen nur Modelle für die englische Sprache an - aus Zeitgründen war es leider nicht möglich, diese Tools umfangreich zu testen, Daten zum Trainieren eines deutschen Modells zu akquirieren und eine Parameteroptimierung durchzuführen. Andererseits wurde festgestellt, dass es bei der Wort-für-Wort-Bearbeitung sehr „natürlich“ ist, Satzzeichen zu ergänzen. Dem Nutzer diese Möglichkeit nicht zur Verfügung zu stellen, würde die User Experience verschlechtern und den Lesefluss sowie das Textverständnis stören.

Abbildung 43: Sonderzeichen-Plug-in

8.2.4 Automatische Großschreibung

Mit diesem Plug-in wird versucht, die fehlende Großschreibung automatisch zu beheben. Beim Drücken des bereitgestellten Toolbar-Buttons wird jedes Wort durchlaufen und mithilfe des HanoverTaggers (Kurz: HanTa) [105] dessen Lemma und Wortklasse extrahiert. Handelt es sich um ein Nomen, wird das Wort automatisch großgeschrieben. Da der HanoverTagger ebenfalls auf ein vortrainiertes Modell setzt, können mit dieser Methode nicht alle Nomen gefunden werden. [44]

8.2.5 DeepSpeech Alternativen

Mithilfe dieses Plug-ins erhält der Nutzer Alternativvorschläge von DeepSpeech, durch die falsche oder fehlende Worte korrigiert werden können. Das Plug-in setzt damit die nicht verpflichtende Anforderung *N00* um. Um dies zu realisieren, werden die Zeitstempel der einzelnen Wörter der Initialtranskription verwendet und anhand dieser das Material erneut mittels DeepSpeech transkribiert. DeepSpeech ist dabei so eingestellt, dass es maximal drei Alternativen liefert. Da dabei aber auch häufig das vorherige oder das nachfolgende Wort vorgeschlagen wird, werden diese aus den Alternativvorschlägen entfernt.

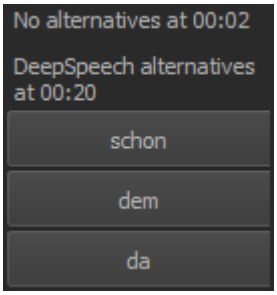


Abbildung 44: DeepSpeech alternativen Plug-in

8.2.6 Vosk Alternativen

Dieses Plug-in ist sehr ähnlich zu dem vorher beschriebenen. Statt DeepSpeech wird hier allerdings Vosk [9] als Speech-To-Text-Engine eingesetzt. Vosk stellt eine Vielzahl von verschiedenen Modellen für unterschiedliche Sprachen bereit. Der Nutzer kann nach eigenem Empfinden kleine (40 MB) oder große (1.2 GB) Modelle verwenden.

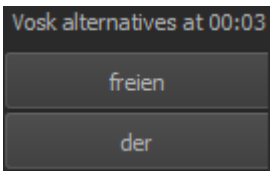


Abbildung 45: Vosk alternativen Plug-in

8.2.7 Umlaute

Dieses Plug-in soll die ausgeschriebenen Umlaute durch den entsprechenden Buchstaben ersetzen. Dafür wird der Text beim Drücken des bereitgestellten Toolbar-Buttons durchlaufen und ausgeschriebene Umlaute ersetzt (siehe Programmausdruck 7). Wörter, bei denen die ausgeschriebenen Umlaute fälschlicherweise ersetzt wurden, können schließlich mittels des LanguageTool-Plug-ins (Kapitel 8.2.8) korrigiert werden.

8.2.8 LanguageTool

Mit dem LanguageTool-Plug-in [60] können die fehlende Großschreibung und andere Rechtschreibfehler korrigiert werden. Dafür muss der Nutzer sich den LanguageTool-Server herunterladen und diesen in das Plug-in Verzeichnis extrahieren. Bei der Initialisierung des Plug-ins wird dieser automatisch gestartet. In der Wort-für-Wort-Bearbeitung wird dann das aktuell markierte Wort an den LanguageTool-Server geschickt und die daraufhin zurückgegebenen Korrekturvorschläge samt Fehlerbeschreibung angezeigt. Der Nutzer kann darüber hinaus Text selbst markieren und über den bereitgestellten Toolbar-Button korrigieren lassen. Das Plug-in setzt damit die nicht verpflichtende Anforderung *N01* um.

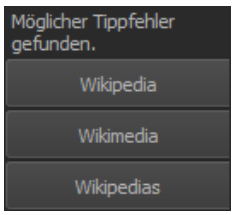


Abbildung 46: LanguageTool Plug-in

8.3 Plug-ins zur Auswertung und Weiterverarbeitung der Transkription

Die nachfolgend vorgestellten Plug-ins dienen der Auswertung und Weiterverarbeitung der Transkription. Sie sind durch die nicht verpflichtenden Anforderungen (Tabelle 6) entstanden.

8.3.1 Zusammenfassung

Mithilfe dieses Plug-ins kann eine einfache Zusammenfassung der Transkription erstellt werden (*N07*). Um die Zusammenfassung zu erzeugen, wird zuerst die Häufigkeit der einzelnen Wörter gezählt. Hierbei werden Füllwörter wie „wir, euer, mir, etc.“ mithilfe der StopWord-Liste des Natural Languge Toolkit [88] ignoriert. Danach wird für jeden Satz die durchschnittliche Häufigkeit pro Wort berechnet und nur die Sätze übernommen, welche einen bestimmten Schwellenwert des Gesamtdurchschnitts überschreiten. Die Länge der Zusammenfassung kann durch Anpassung dieses Schwellenwertes verändert werden. [59], [84]

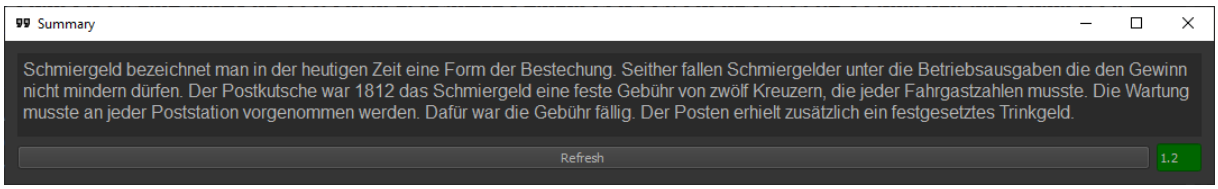


Abbildung 47: Zusammenfassung Plug-in

8.3.2 Anonymisierung

Mithilfe des Fake-Plug-ins ist es möglich, Namen, Adressen, Postleitzahlen oder Telefonnummern durch ausgedachte Werte zu ersetzten und damit die Transkription zu anonymisieren (*N08*). Realisiert wird dies mithilfe der Faker Bibliothek [30], welche es ermöglicht, für unterschiedlichste Typen ausgedachte Werte zu erzeugen. Da die Bibliothek unterschiedliche Sprachen unterstützt, ist eine natürlich wirkende Ersetzung möglich.

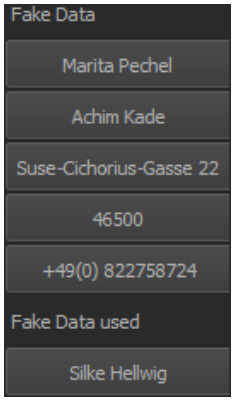


Abbildung 48: Anonymisierung (Fake) Plug-in

8.3.3 Lesbarkeitsindex

Dieses Plug-in zeigt dem Nutzer den aktuellen Flesch-Grad an. Der Flesch-Grad ist ein von Rudolf Flesch entwickeltes Verfahren, welches versucht, die Lesbarkeit eines Textes numerisch zu bewerten (*N03*). Das Verfahren betrachtet dabei die durchschnittliche Satzlänge sowie die durchschnittliche Silbenanzahl pro Wort. Beim Drücken des bereitgestellten Toolbar-Buttons zeigt das Plug-in sowohl den Flesch-Grad, als auch die Einordnung dieses Wertes an. [107]

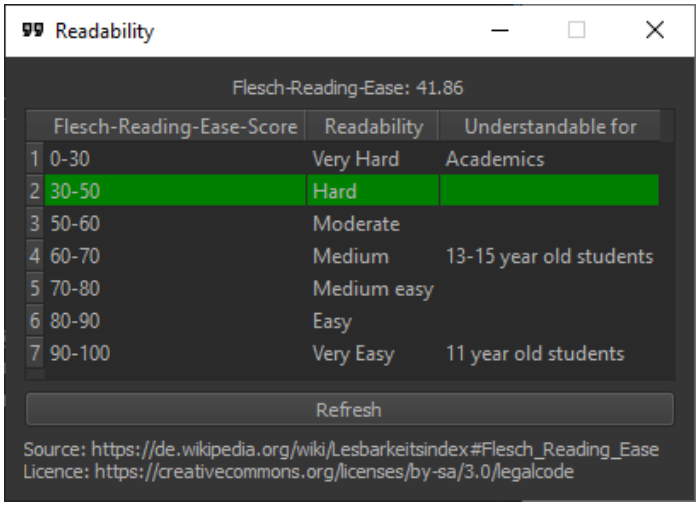


Abbildung 49: Lesbarkeitsindex Plug-in

9 Problem: Forced Alignment

Folgend soll erläutert werden, warum Forced Alignment nicht in LazyTranscript integriert oder implementiert wurde.

Die nicht verpflichtende Anforderung *N06* besagt, dass das System dem Nutzer die Möglichkeit bieten soll, die Transkription mittels Forced Alignment mit dem Gesprochenen zu synchronisieren. Diese Funktionalität würde LazyTranscript in verschiedenen Aspekten deutlich verbessern.

So können aktuell nur Wörter über einen Klick erneut gehört werden, wenn diese in der ersten Transkription vorkamen. Hier werden nämlich durch DeepSpeech Zeitstempel zur Verfügung gestellt, an denen das jeweilige Wort vorkommt. Bei neuen Wörtern hingegen ist es allerdings nicht so leicht, den jeweiligen Zeitstempel herauszufinden oder zu schätzen. Das liegt zum Beispiel daran, dass an anderen Stellen Text entfernt oder ein erkanntes Wort durch mehrere ersetzt wird. Eine mögliche Lösung wäre Forced Alignment, welches bei Textänderungen die Transkription mit dem Gesprochenen synchronisiert und so für jedes Wort einen Zeitstempel zur Verfügung stellt.

Außerdem könnten mittels Forced Alignment automatisch Untertitel-Dateien generiert werden. Hier würde einfach die Transkription vor dem Export synchronisiert und schließlich in das jeweilige Format (wie *.srt* oder *.ass*) konvertiert und exportiert werden.

Mit DSAlign [73] existiert ein Projekt von Mozilla, welches Forced Alignment mittels DeepSpeech umsetzt. Bei Verwendung in LazyTranscript könnten die Modelle für die Transkription also ebenso für das Forced Alignment verwendet werden. Allerdings war es aufgrund verschiedener Fehler nicht möglich, DSAlign zu installieren oder einzusetzen. Aufgrund der zeitlichen Beschränkung wurde auf eine eigene Implementierung des Algorithmus verzichtet.

Andere Forced Alignment Tools haben das Problem, dass sie fast ausschließlich Englisch unterstützen. Da versucht wird so viele Sprachen wie möglich zu unterstützen, sind diese somit ungeeignet. Mit über 30 Sprachen ist „aeneas“ [92] wohl die beste Alternative zu DSAlign. Da für aeneas allerdings noch weitere Programme auf dem System installiert sein müssen, wurde vorerst auf eine Integration verzichtet. [87]

10 Open Source

Für die Veröffentlichung des Projektes auf GitHub wurde die Anleitung „Ein Open-Source-Projekt anfangen“ [25] von Nadia Eghbal, Brandon Keepers, Stephanie Wills und Mike Linksvayer verwendet. Die folgenden Kapitel sollen die wichtigsten Schritte erläutern und zeigen, wie sie für das Projekt umgesetzt wurden. Das Projekt lässt sich unter folgender URL abrufen.

https://github.com/Bettlaken/LazyTranscript

10.1 Lizenz

Damit andere das Projekt ebenfalls nutzen, kopieren oder modifizieren können, sollte eine Open-Source-Lizenz definiert werden. Die beliebtesten Open-Source-Lizenzen sind dabei `MIT`, `Apache 2.0` und `GPLv3`. Da das Projekt in Zukunft für technisch weniger versierte Personen in einem Gesamtpaket ausgeliefert werden soll, muss es unter GPL oder LGPL lizenziert werden. Der Grund dafür liegt bei PySide2. PySide2 kann nur unter GPL oder LGPL lizenziert werden. Diese Lizenzen sind sogenannte „Copyleft-Lizenzen“ und setzen somit unter anderem voraus, dass jegliche Weiterverbreitung ebenfalls unter der gleichen Lizenz geschehen muss. [25], [26], [40], [25]

Für dieses Projekt wurde sich für die GPLv3-Lizenz entschieden, da GNU bei einer einzigartigen Funktionalität die GPL- statt die LGPL-Lizenz empfiehlt. Begründet wird dies damit, dass das Projekt dann nur in freien Programmen verwendet werden kann und somit die Open-Source Gemeinschaft gefördert wird. Eine Veröffentlichung unter der LGPL-Lizenz würde bedeuten, dass eine kommerzielle Nutzung ohne Veröffentlichung des Sourcecode ebenfalls erlaubt wäre. [34]

Zusammengefasst erlaubt die GPL-Lizenz also jegliche Verwendung, solange der Sourcecode veröffentlicht wird und die jeweiligen Lizenzen sowie Veränderungen angegeben werden. Wie bereits erwähnt, müssen die auf diesem Projekt aufbauenden Projekte ebenfalls unter dieser Lizenz veröffentlicht werden. Um die Lizenz anzuwenden, wurde der Lizenztext in der `COPYING.txt`-Datei im Projekt hinterlegt. [39]

10.2 Readme

Im nächsten Schritt galt es, eine Readme-Datei zu erstellen, um andere über das Projekt zu informieren. Hier soll der Leser unter anderem erfahren, worum es in dem Projekt geht, wie er es installieren und verwenden kann und wo er Informationen über das Mitwirken erhält. Im Folgenden soll die nach den Vorschlägen von Danny Gou [42] erstellte Readme genauer erläutert werden. Die vollständige Readme lässt sich in Anhang A.6 in höherer Auflösung einsehen. [25], [42]

Die resultierende Readme kann in vier Teilabschnitte aufgeteilt werden. Der erste Abschnitt (Abbildung 50) soll dem Leser schnell zeigen, worüber das Projekt handelt. Um dieses Ziel zu erreichen, wurde eine kurze Beschreibung verfasst und diese mit einer Grafik der Editor-Maske versehen. Dadurch erfährt der Nutzer einerseits, dass es sich um ein Projekt aus einer Masterarbeit handelt und versucht wird, den Transkriptionsprozess zu vereinfachen und andererseits wie der Editor aussieht und was ihn erwartet.

LazyTranscript

Description

LazyTranscript was developed in a master thesis and tries to simplify the transcription of audio and video material with the help of DeepSpeech and various plug-ins. Through various plug-ins, the editor tries to reduce the effort required to correct the errors in the transcription generated by DeepSpeech.

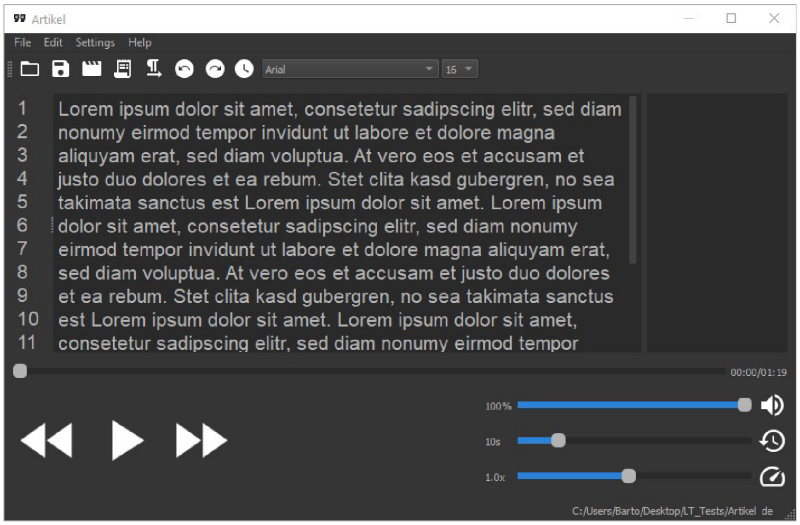


Abbildung 50: Readme: Einführung

Im zweiten Abschnitt (Abbildung 51) erfährt der Leser, wie das Projekt installiert und verwendet werden kann. Um die Installation zu vereinfachen, wurden hierfür die einzelnen Abhängigkeiten in den `requirements.txt`-Dateien festgehalten. Die Dateien beinhalten dabei die Namen der Abhängigkeiten und können mittels des Paketmanagers pip mit einem Befehl installiert werden. Für die Verwendung wird hier ebenfalls auf die Anleitung (Anhang A.5) verwiesen. Etwaige Dateien oder Quellen sind verlinkt und somit klickbar.

Installation and usage

0. Read the manual
1. Install Python 3.8
2. Install the requirements

```
pip install -r requirements.txt
```

3. Remove unwanted plug-ins from the plug-ins directory. Look at the next section for more details about the plug-ins.
4. For each remaining plug-in, run the following command in the respective plug-in folder

```
pip install -r requirements.txt
```

5. Install a DeepSpeech model, see: section 2 in the manual
6. Start LazyTranscript with

```
python main.py
```

7. Edit the Transcription manually or use the word by word editing mode, see manual section 6.5

Abbildung 51: Readme: Installation und Verwendung

Im dritten Abschnitt (Abbildung 52) erfährt der Leser schließlich, welche Plug-ins mit LazyTranscript ausgeliefert werden. Dafür werden diese kurz vorgestellt und etwaige nötige zusätzliche Installationen beschrieben. Zukünftig lohnt es sich hier, für jedes Plug-in eine eigene Readme zu erstellen, um genauere Informationen bereitzustellen.

Preinstalled plug-ins

1. word: This plug-in allows you to prepend, append, capitalize, replace, concat or remove words in the word by word editing mode.
2. words_to_number: This plug-in converts words to the number which they represents. Supports only english and german.
3. add_special_character: This plug-in allows you to append {. , ! ? ( )} to each word in the word by word editing mode.
4. lemmatization: This plug-in adds a toolbar-button which try to capitalize every noun. Currently supports only german.
5. deepspeech_alternatives: This plug-in shows alternative words from deepspeech in the word by word editing mode.
6. vosk_alternatives: This plug-in shows alternative words from vosk in the word by word editing mode. In order to use this plug-in you need to download a model for your language and unzip it in the vosk_alternatives/model folder.
7. umlauts: This plug-in adds a toolbar-button which try to replace all spelled out umlauts
8. language_tool_correction: This plug-in shows correction from a locally running languagetool-server. In order to use this plug-in you need to download the language-server and unzip it in language_tool_correction/language_tool/server. To improve the results, the n-gram models can optionally be installed. They simply have to be unpacked to language_tool/n-gram/<LANGUAGE_TAG>.
9. summary: This plug-in adds a toolbar-button that displays a summary of the current transcription.
10. fake_data: This plug-in shows fake data in the word by word editing mode. This could be useful to remove any personal information.
11. readability: This plug-in adds a toolbar-button that display the current readybility after flesch-reading-ease.

Abbildung 52: Readme: Vorinstallierte Plug-ins

Der letzte Abschnitt (Abbildung 53) beinhaltet hauptsächlich verschiedene Verlinkungen. Hier erfährt der Nutzer wie er Hilfe erhält, welche zukünfigen Pläne existieren und wo er Informationen über das Mitwirken am Projekt findet. Um diesen Schritt einsteigerfreundlicher zu gestalten, wurde außerdem eine Pull Request Anleitung ("PRs welcome") [20] verlinkt. Zuletzt wird hier DANBER für die guten DeepSpeech-Modelle [53] gedankt und auf die Lizenz verlinkt.

Support

If you find any problems or need help, just open an issue.

Roadmap

In the future i would like to implement

1. Forced alignment, to map the transcription to the audio
2. SRT export to create subtitles from the transcription

Contributing

PRs welcome

If you want to make some improvements or create a own plug-in: see CONTRIBUTING.md for more information.

Acknowledgment

Thanks to DANBER for the awesome deepspeech models.

License

This Project is licenced unter GPLv3.

Abbildung 53: Readme: Abschluss

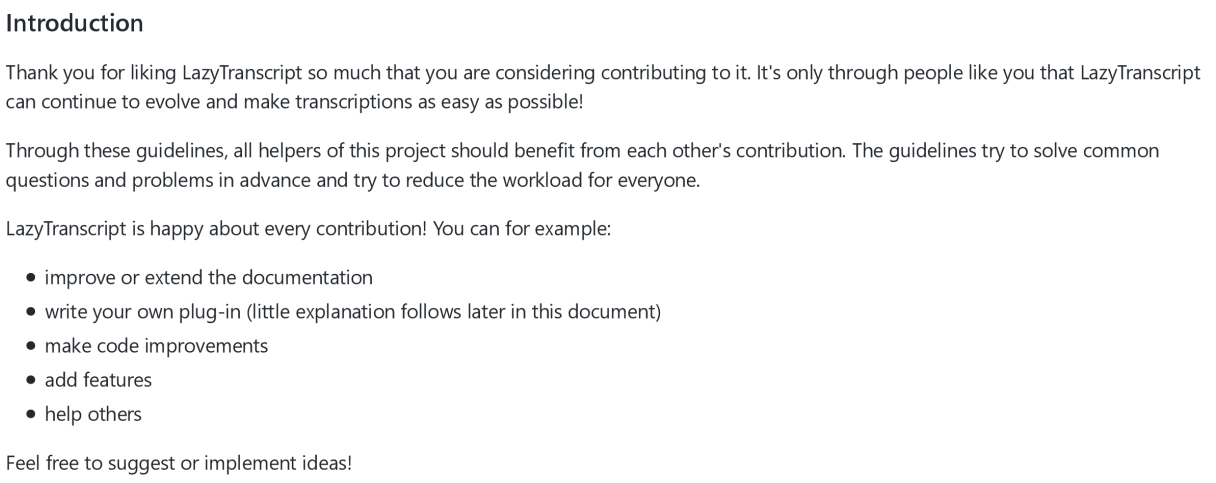
10.3 Contributing

In der Contributing-Datei wird erklärt, wie am Projekt mitgewirkt werden kann. Dafür beschreibt sie unter anderem die folgenden Themen:

- Grund- und Verhaltensregeln
- Wo am Projekt mitgewirkt werden kann
- Wie am Projekt mitgewirkt werden kann
- Wie Fehler gemeldet werden
- Wie Verbesserungen vorgeschlagen werden können [25], [24]

In diesem Fall wurde die Contributing-Datei nach der Vorlage [24] von Nadia Eghbal erstellt und soll im Folgenden genauer erläutert werden. Die vollständige Contributing-Datei lässt sich in Anhang A.7 in höherer Auflösung einsehen.

Im ersten Abschnitt (Abbildung 54) erfolgt eine Einleitung in das Dokument. Dafür wird zuerst erläutert, dass das Dokument versucht, häufige Fragen und Probleme zu beantworten und zu lösen. Außerdem erhält der Leser hier die Information, welchen Beitrag er in diesem Projekt leisten kann. Darunter fällt zum Beispiel die Verbesserung der Dokumentation oder das Schreiben eigener Plug-ins und Features.



Introduction

Thank you for liking LazyTranscript so much that you are considering contributing to it. It's only through people like you that LazyTranscript can continue to evolve and make transcriptions as easy as possible!

Through these guidelines, all helpers of this project should benefit from each other's contribution. The guidelines try to solve common questions and problems in advance and try to reduce the workload for everyone.

LazyTranscript is happy about every contribution! You can for example:

- improve or extend the documentation
- write your own plug-in (little explanation follows later in this document)
- make code improvements
- add features
- help others

Feel free to suggest or implement ideas!

Abbildung 54: Contributing: Einleitung

Im nächsten Abschnitt (Abbildung 55) werden einige Grundregeln aufgestellt. Besonders hervorzuheben ist dabei der von Coraline Ada Ehmke erstellte „Code of Conduct“. Der Code of Conduct ist ein Verhaltenscodex, der beschreibt, wie sich die einzelnen Personen untereinander verhalten sollen und wie mit Regelverstößen im Detail umgegangen werden soll. Beispielsweise soll in der Community ein freundlicher und respektvoller Umgang herrschen. Um dies zu gewährleisten, muss zum Beispiel Feedback konstruktiv geäußert und das Wohl der Community priorisiert werden. Die Konsequenzen für etwaige Regelverstöße sind unter anderem eine öffentliche Entschuldigung, eine Verwarnung oder auch ein (temporärer) Ausschluss. [27]

Unter dem Hinweis zu dem Verhaltenscodex befinden sich des Weiteren noch eigene Regeln. So soll beispielsweise versucht werden, neue Funktionen hauptsächlich als Plug-in zu implementieren. Um etwaige Mergekonflikte zu vermeiden, sollten Features ebenso auf einem eigenen Branch ausgelagert und abgekapselt werden. Außerdem wird empfohlen, Fehler so detailreich wie möglich zu beschreiben und von Fragen bezüglich Python-Grundlagen abzusehen.

Ground Rules

In order to facilitate the collaborative work on LazyTranscript, the following basic rules should be followed.

Follow the CODE_OF_CONDUCT!

- New functionality should preferably be implemented as a plug-in.
- Fill issues not only with error messages, but also with examples.
- The more detailed and extensive the issues are, the easier they are to track.
- Create a new branch for each new feature.
- Features should be as isolated as possible.
- If you are unsure about an idea, feel free to create an issue to discuss your idea and find contributors.
- External programs, which have to be installed without pip, should be avoided urgently. This is to prevent that the installation barrier of LazyTranscript becomes too large.
- Issues should not be used for basic python questions.

Abbildung 55: Contributing: Grundregeln

Nach der Einleitung und den Grundregeln erhält der Nutzer Hilfestellungen (Abbildung 56) zu seinem ersten Beitrag am Projekt. Falls der Leser nicht weiß, wie er helfen soll, wird hier noch einmal auf die Issues hingewiesen. Handelt es sich hierbei um den ersten Pull Request des Lesers, so wird auf die Anleitung „How to Contribute to an Open Source Project on GitHub“ [20] von Kent C. Dodds verwiesen. Um Neueinsteiger noch mehr zu unterstützen, sollen in Zukunft Issues mit einem „first-timers-only“-Tag versehen werden. Diese Issues beinhalten leichte Probleme für Anfänger und sollen Sie motivieren, am Projekt mitzuwirken und an diesem zu lernen. [21]

Your First Contribution

If you want to help with LazyTranscript, but are not sure what to do, it is worth taking a look at the issues. If possible, these are tagged so you can quickly determine that they match your expertise.

Working on your first Pull Request? You can learn how from this *free* series How to Contribute to an Open Source Project on GitHub

Once the first issues are established, it is planned to create and support "first-timers-only" issues according to http://www.firsttimersonly.com/

Now you can start creating your first changes. Feel free to ask for help!

Abbildung 56: Contributing: Erster Beitrag

Der nächste Abschnitt (Abbildung 57) behandelt nun die wichtigsten Informationen zur Implementierung sowie den groben Ablauf. Hier erhält der Leser eine grobe Anleitung des Vorgehens, Hinweise zu (Code-)Konventionen und eine kurze Erläuterung des Reviewprozesses. Sollte es sich um einen kleinen Fehler handeln, wird der Leser dazu motiviert, ein anfängerfreundliches Issue zu erstellen. Da es sich bei dem Projekt in Zukunft um ein Hobbyprojekt handelt, wird der Leser am Schluss darauf hingewiesen, dass voraussichtlich etwas Geduld für die einzelnen Prozesse nötig ist.

Getting started

1. Create your own fork of the code
2. Do the changes in your fork
3. Create a Pull Request

If you find a small fix and don't want to correct it yourself, create a beginner friendly issue so people new to open source can fix them.

Code, commit message and labeling conventions

- Class names should use the CapWords convention
- Function and variable names should be lowercase separated by underscores
- You can find a code documenation in the "docs" folder.

Code review process

The maintainer(s) try to process and comment pull requests as fast as possible. However, do not forget that LazyTranscript is a hobby project and therefore delays are possible.

Abbildung 57: Contributing: Ein Pull Request erstellen

Im vorletzten Abschnitt (Abbildung 58) erfolgt eine kurze Anleitung zur Erstellung eines eigenen Plug-ins. Hier wird überlegt, ob die Dokumentation mit einer übersetzten Version des 8. Kapitels ergänzt werden soll und so die einzelnen Komponenten noch einmal im Detail beleuchtet werden. Außerdem könnte eine komplette Anleitung mit einem Beispiel Plug-in ergänzt werden.

Quick guide for creating your own plug-in

1. Create a folder under the "plugins" directory for your plug-in
2. In this folder create a python-file called "<YOUR_PLUGIN_NAME>_lt_plugin.py"
3. Create a class called "plugin" which inherited the "IPlugin"-Class
4. Override the necessary methods from IPlugin and use the plugin_manager to manipulate or receive the text or specific words
    - If you want to make a toolbar-plug-in then you need override the "get_toolbar_action"-Method and create the QActions there
    - If you want to make a word by word editing plug-in then you need to override the "get_word_action"-Method and return the QPushButtons via the "add_new_word_by_word_action"-Method of the plug-in manager.
5. Override the get_name Method and return your plug-in name

Optional:

- Override the project_loaded-Method to do something when the project is loaded
- For long task you should use a QThread and use the signals in the IPlugin-Class to notify the main thread
- If you are using some open source libraries you should include them in the "licence"-directory in your plug-in directory.

If you have further questions: Have a look at the other plug-ins, read the documentation or feel free to create a issue with your question.

Abbildung 58: Contributing: Ein eigenes Plug-in schreiben

Im letzten Schritt (Abbildung 59) wird erläutert, wie Fehler oder Features gemeldet beziehungsweise vorgeschlagen werden sollen. Grundsätzlich sollten diese so detailreich wie möglich beschrieben werden. Dies erleichtert die Fehler- und Machbarkeitsanalyse erheblich. Sollte in Zukunft deutlich werden, dass bei jeder Fehlermeldung gewisse Standardinformationen nötig sind, so soll hier ein extra Template erstellt und verwendet werden.

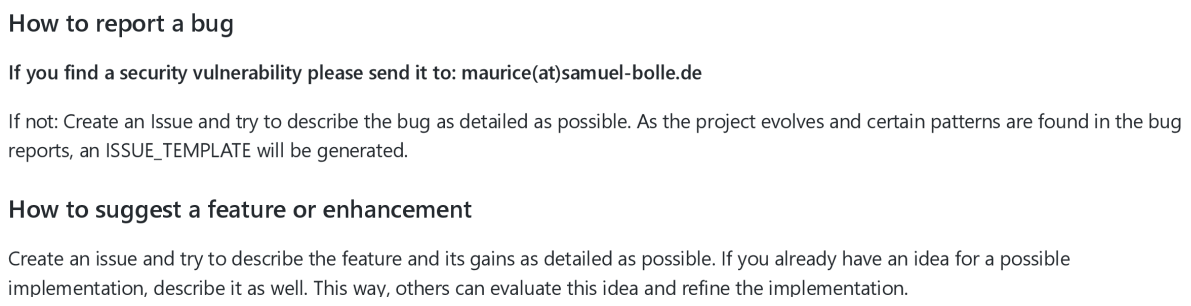


Abbildung 59: Contributing: Fehler und Features vorschlagen oder melden

10.4 Projektname

Im nächsten Schritt gilt es den Projektnamen zu definieren. An diesen werden nach der Anleitung [25] drei wichtige Anforderungen gestellt:

1. Der Projektname sollte leicht zu merken sein.
2. Der Projektname sollte, wenn möglich, eine grobe Vorstellung über das Projekt geben.
3. Der Projektname sollte nicht aus Wortspielen oder Wortwitzen bestehen, welche von anderen Menschen nicht gut übersetzt werden können.

Wie im Dokument an verschiedenen Stellen deutlich wurde, fiel die Entscheidung auf „LazyTranscript“. Der Name soll verdeutlichen, dass das Projekt dabei hilft, Transkriptionen mit weniger Aufwand zu erstellen. Nach dem Open Source Project Name Checker [81], der Instant Domain Search [52] sowie der WIPO Global Brand Database [111] wird der Name weder in einem Projekt, noch als Domain oder sonstige Marke verwendet (Stand 06.01.2021).

10.5 Code

Die Dokumentation des Codes mittels Docstrings erfolgt im Google Style. Bei diesem werden für Funktionen beispielsweise eine Kurzbeschreibung und eine optionale längere Beschreibung verfasst sowie die Parameter und der Rückgabewert beschrieben. Ein Beispiel lässt sich in der napoleon-Dokumentation [93] oder im oben genannten Repository von LazyTranscript einsehen. In Zukunft könnte dieser DocString mit einer der vielen verschiedenen Tools in eine webbasierte Lösung transformiert und dort dargestellt werden.

11 Zusammenfassung und Fazit

Im ersten Teil dieser Arbeit wurde verdeutlicht, warum ein auf DeepSpeech basierender Open Source Editor zur Transkription nötig ist. So existieren aktuell in diesem Bereich zwei Gruppen von Programmen und Diensten. Die eine Gruppe versucht den Nutzer bei der Transkription durch Annehmlichkeiten, wie beispielsweise Tastenkombinationen oder Textbausteinen, zu unterstützen. Dabei muss der Nutzer den Text allerdings immer noch selbst abtippen. Die andere Gruppe führt hingegen die Transkription automatisch durch. Problematisch hierbei sind jedoch die (für Privatpersonen) hohen Kosten sowie die datenschutzrelevanten Bedenken bei der Verwendung von Online-Diensten wie beispielsweise f4x oder Google Cloud Speech-To-Text. Es wurde erläutert, dass deshalb im Rahmen dieser Arbeit ein Editor entwickelt wird, welcher eine automatische Transkription offline bereitstellt und dem Nutzer zusätzlich die Möglichkeit bietet, diese mit Hilfe von verschiedenen Plug-ins zu korrigieren.

Um die Funktionsweise und Besonderheiten von DeepSpeech hervorzuheben, wurde danach die klassische Spracherkennung erläutert und beleuchtet. Die wichtigsten Komponenten eines klassischen Spracherkennungssystems sind dabei ein Hidden Markov Modell zur Bestimmung der wahrscheinlichsten Phone, ein Wörterbuch, um aus den Phonem-Kombinationen die richtigen Wörter zu finden und ein Sprachmodell, um das wahrscheinlichste Wort zu erkennen. Erschwert wird der Spracherkennungsprozess dabei besonders durch die Mehrdeutigkeit der Sprache sowie der Größe des zu erkennenden Wortschatzes. Außerdem sind die Aussprache der einzelnen Personen und die Aufnahmeumgebung entscheidende Faktoren, wie gut eine Spracherkennung funktioniert.

Danach erfolgte die Vorstellung der Speech-To-Text-Engine DeepSpeech von Mozilla. Hier wurde erläutert, dass Mozilla mit DeepSpeech eine offene Speech-To-Text-Engine entwickeln wollte, welche durch einen Machine Learning Ansatz verschiedene Sprachen mit einer hohen Ergebnisqualität unterstützt. Die Bereitstellung verschiedener Sprachmodelle wie auch die Einschränkungen anderer Speech-To-Text-Engines haben schließlich dazu geführt, diese Arbeit auf DeepSpeech aufzubauen.

Im Gegensatz zur klassischen Spracherkennung werden bei DeepSpeech Buchstaben durch ein neuronales Netz erkannt und mittels des Connectionist Temporal Classification-Algorithmus zu verschiedenen Wörtern zusammengefasst. Mithilfe eines Sprachmodells wird anschließend ebenso das wahrscheinlichste Wort erkannt.

Auf die bisherigen Erkenntnisse erfolgte aufbauend die Konzeption des Editors. Hier wurden die Anforderungen an den zu entwickelnden Editor definiert. Diese wurden mit Hilfe eines Brainstormings sowie verschiedener Tests bereits existierenden Transkriptionsprogramme erhoben. Diese Anforderungen wurden in zwei Kategorien aufgeteilt. Die verpflichtenden An-

forderungen beinhalten die Grundfunktionen des Editors. Dazu gehören zum Beispiel das Bereitstellen einer Transkription, das Abspielen von Audio- und Videomaterial oder auch das Anpassen der Schriftgröße. Die nicht verpflichtenden Anforderungen beinhalten Funktionen, die größenteils als Plug-in realisiert werden sollten. Darunter zum Beispiel Alternativvorschläge durch andere Speech-To-Text-Engines oder auch die Funktion Satzzeichen wiederherzustellen.

Auf Basis dieser Anforderungen wurden anschließend die Wireframes erstellt. Hier wurde das grobe Aussehen sowie die grundsätzliche Funktionsweise des Editors definiert. Neben der klassischen Funktionsweise eines Texteditors wurde hier auch festgelegt, wie der Korrekturprozess durch das Hervorheben und Bearbeiten einzelner Wörter funktionieren soll.

Im ersten Teil der Implementierung wurden dann die verpflichtenden Anforderungen mittels PySide2 in Python umgesetzt und vorgestellt. Um die nötigen Plug-ins zu entwickeln, die dem Nutzer die Möglichkeit bieten, die von DeepSpeech getätigte Transkription zu korrigieren, wurde daraufhin ein Beispieltext untersucht. Es hat sich gezeigt, dass in der Transkription von DeepSpeech die Großschreibung, die Satzzeichen, Umlaute oder auch ganze Wörter fehlen. Mithilfe der daraufhin erläuterten Plug-in-Struktur, wurden zur Behebung dieser Fehler verschiedene Plug-ins implementiert und vorgestellt. Sie ermöglichen dem Nutzer die folgenden Funktionen:

- Bearbeitung (Anhängen, Ersetzen, Entfernen, etc.) von einzelnen Wörtern
- Konvertierung von Wörtern zu Zahlen
- Anhängen verschiedener Satzzeichen
- Automatische Berichtigung der fehlenden Großschreibung durch Lemmatisierung
- Ersetzung der Wörter durch Alternativvorschläge von den Speech-To-Text-Engines DeepSpeech und Vosk
- Automatische Berichtigung der ausgeschriebenen Umlaute
- Rechtschreibkorrektur durch das LanguageTool

Die Menge der Plug-ins wurde zusätzlich durch verschiedene Plug-ins zur Weiterverarbeitung und Auswertung der Transkription erweitert. Der Nutzer kann sich damit eine einfache Zusammenfassung der Transkription ansehen, die Transkription mittels ausgedachter Daten anonymisieren oder die Lesbarkeit mithilfe des Flesch-Grads überprüfen.

Insgesamt wurden alle verpflichtenden Anforderungen umgesetzt. Neben klassischen Funktionen von Textverarbeitungsprogrammen, wie das Anpassen der Schriftart oder dem Anzeigen einer Zeilennummerierung, verfügt der Editor auch über spezielle Funktionen hinsichtlich des

Transkribierens. Darunter das Erstellen einer Initialtranskription mittels DeepSpeech, das Abspielen und Steuern von Audio- und Videomaterial sowie die Verwendung von Textbausteinen und Tastenkombinationen. Von den zehn nicht verpflichtenden Anforderungen wurden vier aufgrund der begrenzten Zeit nicht implementiert. Diese werden im Ausblick (Kapitel 12) noch einmal genauer behandelt. Die implementierten entsprechen den bereits vorgestellten Plug-ins zur Korrektur und Weiterverarbeitung der Transkription.

Für die Veröffentlichung des Projektes wurde die GPLv3-Lizenz ausgewählt. Außerdem wurde eine Readme- und Contributing-Datei angelegt, der Code kommentiert sowie der Name für die im Rahmen dieser Arbeit entwickelten Applikation auf LazyTranscript festgelegt.

Zusammenfassend lässt sich festhalten, dass die Konzeption und Entwicklung eines auf DeepSpeech basierenden Open Source Editors zur Transkription von Audio- und Videomaterial erfolgreich durchgeführt wurde. Im Gegensatz zu den in der Einleitung vorgestellten Transkriptionsprogrammen und Transkriptionsdiensten bietet der hier entwickelte Editor eine kostenfreie Offline-Alternative. Der Nutzer muss dabei nicht die komplette Transkription selbst durchführen, sondern erhält durch die verschiedenen Plug-ins Unterstützung, um die von DeepSpeech bereitgestellte Transkription leicht und schnell zu korrigieren.

12 Ausblick

Das Projekt bietet, besonders aufgrund der begrenzten Zeit, einige Möglichkeiten zur Verbesserung oder Fortführung. Beispielsweise könnte in Zukunft die Dokumentation weiter ausgebaut werden. Eine Möglichkeit wäre zum Beispiel das Erstellen eines eigenen Wikis. Hier könnten detaillierte Beschreibungen und Anleitungen zu den bereits implementierten Plug-ins oder zur Implementierung von neuen Plug-ins angelegt werden. Außerdem wäre es möglich die Contributing-Datei weiter auszubauen. So könnte der Pull-Request-Prozess oder die Erstellung von Issues genauer spezifiziert werden.

Natürlich könnte und sollte LazyTranscript durch die Implementierung weiterer Plug-ins erweitert werden. Wie in Kapitel 9 erläutert wurde, wäre es sinnvoll, ein Plug-in zu implementieren, welches mittels Forced Alignment die Transkription auf das Gesprochene synchronisiert (*N06*). Damit könnte eine einfache Möglichkeit realisiert werden, die Transkription ebenso automatisch als Untertitel zu exportieren (*N09*). Des Weiteren könnte ein Plug-in implementiert werden, welches eine Sprechererkennung durchführt und dem Nutzer anbietet, die Ergebnisse in die Transkription zu integrieren (*N10*).

Neben der Implementierung von neuen Plug-ins könnten die bisherigen aber auch erweitert werden. Eine Möglichkeit wäre eine automatische Wiederherstellung der Satzzeichen mittels des Punctuators [97]. Für die Erstellung eines Modells müsste dafür unter anderem untersucht werden, welche Anforderungen an Trainingsdaten bestehen und wie die Trainingsdaten gefunden oder erzeugt werden können.

Grundsätzlich wurde LazyTranscript mit Hilfe von Python plattformunabhängig implementiert. Da es aber nur auf einem Windows Rechner getestet wurde, gilt es zu überprüfen, ob alle gängigen Betriebssysteme vollständig und korrekt unterstützt werden und ob alle möglichen Fehlerfälle korrekt behandelt oder betrachtet wurden.

LazyTranscript bietet aber auch Raum für neue Funktionen. Um das (De-)Installieren von Plug-ins noch leichter zu gestalten, wäre es möglich dafür eine eigene Maske zu implementieren. Außerdem könnte die Funktionalität umgesetzt werden, Audio- oder Videomaterial vor Erstellen der Transkription zu schneiden (*N04*). Dadurch wäre der Nutzer fähig, nicht für die Transkription gewollte Stellen von Beginn an zu entfernen. Ein mögliches größeres Feature könnte die Integration von Continual Learning sein. Nach [58, S.1] beschreibt Continual Learning die Fähigkeit, sich kontinuierlich neues Wissen aus zuvor Erlerntem anzueignen. Im Fall von LazyTranscript könnte also die Korrektur der von DeepSpeech bereitgestellten Transkription wieder zurück in das Modell fließen. Es wäre möglich, dass sich so das Modell an den jeweiligen Nutzer und dessen Transkriptionen anpasst und damit die automatischen Transkriptionen mit der Zeit besser werden.

Referenzen

[1] Aashish Agarwal und Torsten Zesch. „German End-to-end Speech Recognition based on DeepSpeech“. In: *Preliminary proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers*. Erlangen, Germany: German Society for Computational Linguistics & Language Technology, 2019, S. 111–119. URL: https://www.researchgate.net/publication/336532830_German_End-to-end_Speech_Recognition_based_on_DeepSpeech (besucht am 23. 09. 2020).

[2] Explosion AI. *Industrial-Strength Natural Language Processing*. URL: https://spacy.io/ (besucht am 21. 10. 2020).

[3] AlphaCephei. *Update on CMUSphinx Project*. Okt. 2019. URL: https://cmusphinx.github.io/2019/10/update/ (besucht am 28. 09. 2020).

[4] AmberScript. *Preise*. URL: https://www.amberscript.com/de/preise#top (besucht am 27. 11. 2020).

[5] audiotranskription. *f4transkript - Manuell transkribieren*. URL: https://www.audiotranskription.de/f4 (besucht am 01. 10. 2020).

[6] audiotranskription. *f4x - Automatische Spracherkennung*. URL: https://www.audiotranskription.de/f4x (besucht am 01. 10. 2020).

[7] audiotranskription. *f4x Spracherkennung - Kontingent*. URL: https://www.audiotranskription.de/shop/de/f4x.html (besucht am 27. 11. 2020).

[8] Alpha Cephei. *OPEN SOURCE SPEECH RECOGNITION TOOLKIT*. URL: https://cmusphinx.github.io/ (besucht am 26. 09. 2020).

[9] Alpha Cephei. *Vosk*. URL: https://alphacephei.com/vosk/ (besucht am 26. 09. 2020).

[10] Fausto Cercignani. *Beispielsätze mit deutschen Homophonen*. 2020. URL: https://sites.unimi.it/austheod/FC_DEUTSCHE_HOMOPHONE.htm (besucht am 18. 09. 2020).

[11] Google Cloud. *Speech-to-Text*. URL: https://cloud.google.com/speech-to-text?hl=de (besucht am 24. 09. 2020).

[12] Google Cloud. *Speech-to-Text - Preise*. URL: https://cloud.google.com/speech-to-text/pricing?hl=de (besucht am 27. 11. 2020).

[13] Creative Commons. *Attribution-ShareAlike 3.0 Unported*. URL: https://creativecommons.org/licenses/by-sa/3.0/legalcode (besucht am 17. 11. 2020).

[14] Creative Commons. *Attribution-ShareAlike 4.0 International*. URL: https://creativecommons.org/licenses/by-sa/4.0/legalcode (besucht am 17. 11. 2020).

[15] Riverbank Computing. *What is PyQt?* URL: https://www.riverbankcomputing.com/software/pyqt/ (besucht am 22. 10. 2020).

[16] Arne Cypionka. *„Spracherkennung muss jedem dienen, unabhängig von seiner Wirtschaftskraft!“* 2018. URL: https://netzpolitik.org/2018/spracherkennung-muss-jedem-dienen-unabhaengig-von-seiner-wirtschaftskraft (besucht am 23. 09. 2020).

[17] Kelly Davis. *Deep Speech: Free(ing) Speech with Deep Learning*. Hochgeladen 2019. URL: https://www.youtube.com/watch?v=ZDgHS0wTYuo (besucht am 23. 09. 2020).

[18] DictaNet. *Online Store*. URL: https://www.dictanet.com/dictanet-pc-spracherkennung.html (besucht am 01. 10. 2020).

[19] DictaNet. *Online Store*. URL: https://store.jurasoft.de/rae_net/store/index.aspx?showStore=1&bereich=1&produktgruppe=3&zeigen=-1&login=1 (besucht am 27. 11. 2020).

[20] Kent C. Dodds. *How to Contribute to an Open Source Project on GitHub*. URL: https://egghead.io/courses/how-to-contribute-to-an-open-source-project-on-github (besucht am 06. 01. 2021).

[21] Kent C. Dodds und Scott Hanselman. *First Timers Only*. URL: https://www.firsttimersonly.com/ (besucht am 06. 01. 2021).

[22] Mike Driscoll. „wxPython: Creating a Simple Media Player“. In: (2010). URL: https://www.blog.pythonlibrary.org/2010/07/24/wxpython-creating-a-simple-media-player/ (besucht am 21. 10. 2020).

[23] Duden. *Rechtschreibung gestern und heute*. 2020. URL: https://www.duden.de/ueber_duden/geschichte-der-rechtschreibung (besucht am 18. 09. 2020).

[24] Nadia Eghbal. *CONTRIBUTING-template.md*. URL: https://github.com/nayafia/contributing-template/blob/HEAD/CONTRIBUTING-template.md (besucht am 03. 01. 2021).

[25] Nadia Eghbal u. a. *Ein Open-Source-Projekt anfangen*. URL: https://opensource.guide/de/starting-a-project/ (besucht am 03. 01. 2021).

[26] Nadia Eghbal u. a. *Rechtliche Aspekte von Open-Source-Projekten*. URL: https://opensource.guide/de/legal/ (besucht am 03. 01. 2021).

[27] Coraline Ada Ehmke. *Contributor Covenant*. URL: https://www.contributor-covenant.org/ (besucht am 06. 01. 2021).

[28] Karl Eilebrecht und Gernot Starke. *Patterns kompakt. Entwurfsmuster für effektive Software-Entwicklung*. 4. Aufl. Berlin, Heidelberg: Springer Vieweg, 2013.

[29] Stephen Euler. *Grundkurs Spracherkennung. Vom Sprachsignal zum Dialog — Grundlagen und Anwendung verstehen — Mit praktischen Übungen.* 1. Aufl. Wiesbaden: Friedr.Vieweg & Sohn Verlag, GWV Fachverlage GmbH, 2006.

[30] Daniele Faraglia. *GitHub - Faker*. URL: https://github.com/joke2k/faker (besucht am 26. 12. 2020).

[31] FFmpeg. *FFmpeg - Homepage*. URL: https://ffmpeg.org/ (besucht am 15. 11. 2020).

[32] Martin Fitzpatrick. *PyQt5 vs PySide2: What's the difference between the two Python Qt libraries?* URL: https://www.learnpyqt.com/blog/pyqt5-vs-pyside2/ (besucht am 22. 10. 2020).

[33] Free Software Foundation. *QMediaPlayer*. URL: https://doc.qt.io/qtforpython/PySide2/QtMultimedia/QMediaPlayer.html (besucht am 22. 10. 2020).

[34] Free Software Foundation. *Why you shouldn't use the Lesser GPL for your next library*. URL: https://www.gnu.org/licenses/why-not-lgpl.en.html (besucht am 22. 10. 2020).

[35] Python Software Foundation. *importlib - The implementation of import*. URL: https://docs.python.org/3/library/importlib.html (besucht am 21. 12. 2020).

[36] Qt Foundation. *Licensing*. URL: https://www.qt.io/licensing/ (besucht am 22. 10. 2020).

[37] Qt Foundation. *One framework. One codebase. Any platform.* URL: https://www.qt.io/ (besucht am 22. 10. 2020).

[38] Mark Gales und Steve Young. „The Application of Hidden Markov Models in Speech Recognition". In: *Foundations and Trends in Signal Processing* 1.3 (2007). URL: https://mi.eng.cam.ac.uk/~mjfg/mjfg_NOW.pdf (besucht am 19. 09. 2020).

[39] GitHub. *GNU General Public License v3.0*. URL: https://choosealicense.com/licenses/gpl-3.0/ (besucht am 03. 01. 2021).

[40] GNU. *Copyleft. Was ist das?* URL: https://www.gnu.org/licenses/#WhatIsCopyleft (besucht am 03. 01. 2021).

[41] Joachim Goll und Manfred Dausmann. *Architektur- und Entwurfsmuster der Softwaretechnik. Mit lauffähigen Beispielen in Java*. Wiesbaden: Springer Vieweg, 2013.

[42] Danny Guo. *Suggestions for a good README*. URL: https://www.makeareadme.com/#suggestions-for-a-good-readme (besucht am 03. 01. 2021).

[43] Satish Chandra Gupta. *How to build Python transcriber using Mozilla DeepSpeech*. 2020. URL: https://www.slanglabs.in/blog/how-to-build-python-transcriber-using-mozilla-deepspeech (besucht am 29. 09. 2020).

[44] Hochschule Hannover. *Vorverarbeitung von Texten mit Python und NLTK*. URL: https://textmining.wp.hs-hannover.de/Preprocessing.html (besucht am 25. 12. 2020).

[45] Awni Hannun. „Sequence Modeling with CTC“. In: *Distill* (2017). DOI: 10.23915/distill.00008. URL: https://distill.pub/2017/ctc (besucht am 23. 09. 2020).

[46] Awni Hannun. *Stanford Seminar - Deep Speech: Scaling up end-to-end speech recognition*. 2015. URL: https://www.youtube.com/watch?v=P9GLDezYVX4 (besucht am 21. 09. 2020).

[47] Awni Hannun u. a. *Deep Speech: Scaling up end-to-end speech recognition*. 2014. URL: https://arxiv.org/abs/1412.5567 (besucht am 23. 09. 2020).

[48] Peter Flock und Helge Spieker. *Spracherkennung*. Hochschule Bonn-Rhein-Sieg. 2017. URL: https://www.h-brs.de/files/20171215_fbinf_mclab_ss15_spracherkennung_flock_spieker_sa_mk.pdf (besucht am 18. 09. 2020).

[49] IBM. *GitHub - Word to Number*. URL: https://github.com/IBM/wort-to-number (besucht am 25. 12. 2020).

[50] IBM. *Watson Speech to Text*. URL: https://www.ibm.com/de-de/cloud/watson-speech-to-text (besucht am 24. 09. 2020).

[51] IBM. *Watson Speech to Text - Preisgestaltung*. URL: https://www.ibm.com/de-de/cloud/watson-speech-to-text/pricing (besucht am 27. 11. 2020).

[52] Inc. Instant Domain Search. *Instant Domain Search*. URL: https://instantdomainsearch.com/ (besucht am 06. 01. 2021).

[53] Jaco-Assistant. *GitLab - DeepSpeech Polyglot*. URL: https://gitlab.com/Jaco-Assistant/deepspeech-polyglot (besucht am 28. 09. 2020).

[54] Kaldi-asr. *GitHub - Kaldi Speech Recognition Toolkit*. URL: https://github.com/kaldi-asr/kaldi (besucht am 28. 09. 2020).

[55] Kaldi-asr. *Homepage*. URL: http://kaldi-asr.org/ (besucht am 28. 09. 2020).

[56] Tilman Kamp. *Mozilla's DeepSpeech and Common Voice projects*. Vortrag auf der FOSDEM 2018, Folien zu finden unter https://tilmankamp.github.io/FOSDEM2018/. 2018. URL: https://www.youtube.com/watch?v=NtZipf0BxKg (besucht am 23. 09. 2020).

[57] Matthias Senke (KnubellTTV). *Datei:De-Schmiergeld-article.ogg*. URL: https://de.wikipedia.org/wiki/Datei:De-Schmiergeld-article.ogg (besucht am 17. 11. 2020).

[58] Eric Koepke u. a. *Continuous Learning of Deep NeuralNetworks*. 2019. URL: https://www.di-lab.tum.de/fileadmin/w00byz/www/Precibake_-_TUM-DI-LAB_Final_Documentation_SS19.pdf (besucht am 11. 01. 2021).

[59] Kanishka Lahori. *Python - Text Summarizer*. URL: https://www.geeksforgeeks.org/python-text-summarizer/ (besucht am 26. 12. 2020).

[60] LanguageTool. *LanguageTool - Grammatik- und Rechtschreibprüfung*. URL: https://languagetool.org/de/ (besucht am 21. 10. 2020).

[61] Gerard Marull-Paretas. *GitHub - qtmodern*. URL: https://github.com/gmarull/qtmodern (besucht am 15. 11. 2020).

[62] Benjamin Milde und Arne Köhn. *GitHub - Open source speech recognition recipe and corpus for building German acoustic models with Kaldi*. URL: https://github.com/uhh-lt/kaldi-tuda-de#pretrained-models (besucht am 28. 09. 2020).

[63] Julian Moeser. *Künstliche neuronale Netze – Aufbau & Funktionsweise*. 2018. URL: https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/ (besucht am 24. 09. 2020).

[64] Reuben Morais. *A Journey to <10% Word Error Rate*. 2017. URL: https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/ (besucht am 23. 09. 2020).

[65] Reuben Morais. *Speech-To-Text And Back - Open Source Speech Technology at Mozilla*. Vortrag auf der AI Assistant Summit. 2019. URL: https://videos.re-work.co/videos/1708-speech-to-text-and-back-open-source-speech-technology-at-mozilla (besucht am 23. 09. 2020).

[66] Reuben Morais. *Streaming RNNs in TensorFlow*. 2018. URL: https://hacks.mozilla.org/2018/09/speech-recognition-deepspeech/ (besucht am 24. 09. 2020).

[67] Mozilla. *Common-Voice*. URL: https://commonvoice.mozilla.org/de (besucht am 23. 09. 2020).

[68] Mozilla. *Common-Voice: Sprachen*. URL: https://commonvoice.mozilla.org/de/languages (besucht am 23. 09. 2020).

[69] Mozilla. *Common-Voice: Über uns*. URL: https://commonvoice.mozilla.org/de/about (besucht am 23. 09. 2020).

[70] Mozilla. *DeepSpeech Model*. URL: https://deepspeech.readthedocs.io/en/v0.8.0/DeepSpeech.html (besucht am 23. 09. 2020).

[71] Mozilla. *GitHub - DeepSpeech-examples*. URL: https://github.com/mozilla/DeepSpeech-examples/tree/ (besucht am 15. 11. 2020).

[72] Mozilla. *GitHub - DeepSpeech-examples - vad_transcriber*. URL: https://github.com/mozilla/DeepSpeech-examples/tree/r0.8/vad_transcriber (besucht am 04. 10. 2020).

[73] Mozilla. *GitHub - DSAlgin*. URL: https://github.com/mozilla/DSAlign (besucht am 30. 12. 2020).

[74] Mozilla. *GitHub - Project DeepSpeech*. URL: https://github.com/mozilla/DeepSpeech (besucht am 23. 09. 2020).

[75] Mozilla. *Python API Usage example*. URL: https://deepspeech.readthedocs.io/en/v0.8.2/Python-Examples.html (besucht am 29. 09. 2020).

[76] Mozilla. *Welcome to DeepSpeech's documentation!* URL: https://deepspeech.readthedocs.io/en/latest/?badge=latest (besucht am 23. 09. 2020).

[77] Akshay Nagpal. *GitHub - Word to Number*. URL: https://github.com/akshaynagpal/w2n (besucht am 25. 12. 2020).

[78] Nuance. *Spracherkennung Dragon Home Version 15*. URL: https://www.nuance.com/de-de/dragon/dragon-for-pc/home-edition.html (besucht am 01. 10. 2020).

[79] Linguistik Online. *Pragmatik*. 2009. URL: https://wiki.uni-due.de/LinguistikOnline/index.php/Pragmatik (besucht am 19. 09. 2020).

[80] Linguistik Online. *Semantik*. 2009. URL: https://wiki.uni-due.de/LinguistikOnline/index.php/Semantik (besucht am 19. 09. 2020).

[81] Ospnc. *Open Source Project Name Checker*. URL: http://ivantomic.com/projects/ospnc/ (besucht am 06. 01. 2021).

[82] oTranscribe. *Github - oTranscribe*. URL: https://github.com/oTranscribe/oTranscribe (besucht am 01. 10. 2020).

[83] oTranscribe. *oTranscribe Help*. URL: https://otranscribe.com/help/ (besucht am 05. 10. 2020).

[84] Akash Panchal. *Text summarization in 5 steps using NLTK: WordFrequency Algorithm*. URL: https://becominghuman.ai/text-summarization-in-5-steps-using-nltk-65b21e352b65 (besucht am 26. 12. 2020).

[85] Pietro Passarelli. *autoEdit 3*. URL: https://www.autoedit.io/ (besucht am 06. 10. 2020).

[86] Pietro Passarelli. *autoEdit 3 - Speech to text options*. URL: https://autoedit.gitbook.io/autoedit-3-user-manual/speech-to-text (besucht am 07. 10. 2020).

[87] Alberto Pettarin. *GitHub - forced-alignment-tools*. URL: https://github.com/pettarin/forced-alignment-tools (besucht am 30. 12. 2020).

[88] NLTK Project. *Natural Language Toolkit*. URL: https://www.nltk.org/ (besucht am 21. 10. 2020).

[89] PySimpleGui. *GitHub - Demo Media Player VLC Based*. URL: https://github.com/PySimpleGUI/PySimpleGUI/blob/master/DemoPrograms/Demo_Media_Player_VLC_Based.py (besucht am 21. 10. 2020).

[90] Python-Wiki. *GUI Programming in Python*. URL: https://wiki.python.org/moin/GuiProgramming (besucht am 21. 10. 2020).

[91] Qt-Wiki. *Qt for Python*. URL: https://wiki.qt.io/Qt_for_Python (besucht am 22. 10. 2020).

[92] ReadBeyond. *aeneas*. URL: https://www.readbeyond.it/aeneas/ (besucht am 30. 12. 2020).

[93] Rob Ruan. *Example Google Style Python Docstrings*. URL: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html (besucht am 08. 01. 2021).

[94] Chris Rupp und die Sophisten. *Requirements-Engineering und -Mangagement. Aus der Praxis von klassisch bis agil*. 6. Aufl. München: Carl Hanser Verlag, 2014.

[95] Surya Pratap Singh. *https://iq.opengenus.org/fully-connected-layer/*. URL: https://iq.opengenus.org/fully-connected-layer/ (besucht am 24. 09. 2020).

[96] Julius Speech. *GitHub - Julius: Open-Source Large Vocabulary Continuous Speech Recognition Engine*. URL: https://github.com/julius-speech/julius (besucht am 28. 09. 2020).

[97] Chris Spencer. *GitHub - Punctuator*. URL: https://github.com/chrisspen/punctuator2 (besucht am 11. 01. 2021).

[98] Axel Susen. *Spracherkennung. Kosten, Nutzen, Einsatzmöglichkeiten*. Berlin, Offenbach: VDE-Verlag GmbH, 1999.

[99] Alexander Veysov (Silero AI Team). *GitHub - Silero Models*. URL: https://github.com/snakers4/silero-models (besucht am 28. 09. 2020).

[100] Silero AI Team. *Silero Speech-To-Text Models*. URL: https://pytorch.org/hub/snakers4_silero-models_stt/ (besucht am 28. 09. 2020).

[101] Beat Pfister und Tobias Kaufmann. *Sprachverarbeitung. Grundlagen und Methoden der Sprachsynthese und Spracherkennung*. 2. Aufl. Berlin, Heidelberg: Springer Vieweg., 2017.

[102] ubuntuusers. *Transkriptionsprogramme*. URL: https://wiki.ubuntuusers.de/Transkriptionsprogramme/ (besucht am 05. 10. 2020).

[103] Verschiedene. *Links to pretrained models*. URL: https://discourse.mozilla.org/t/links-to-pretrained-models/62688 (besucht am 28. 09. 2020).

[104] VoxForge. *Downloads*. URL: http://www.voxforge.org/de/downloads (besucht am 28. 09. 2020).

[105] Christian Wartena. „A Probabilistic Morphology Model for German Lemmatization“. In: Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019). 2019, S. 40 –49. URL: http://nbn-resolving.de/urn:nbn:de:bsz:960-opus4-15271.

[106] E Werkzeug. *easytranscript*. URL: https://www.e-werkzeug.eu/index.php/de/produkte/easytranscript (besucht am 01. 10. 2020).

[107] Wikipedia. *Lesbarkeitsindex*. URL: https://de.wikipedia.org/wiki/Lesbarkeitsindex#Flesch_Reading_Ease (besucht am 26. 12. 2020).

[108] Wikipedia. *Portal:Gesprochene Wikipedia/Liste*. URL: https://de.wikipedia.org/wiki/Portal:Gesprochene_Wikipedia/Liste (besucht am 17. 11. 2020).

[109] verschiedene Wikipedia. *Schmiergeld*. URL: https://de.wikipedia.org/wiki/Schmiergeld (besucht am 17. 11. 2020).

[110] R. Wilke. *Was ist, wie funktioniert und wozu dient Spracherkennung?* 2010. URL: http://dragon-spracherkennung.forumprofi.de/t20f3-Was-ist-wie-funktioniert-und-wozu-dient-Spracherkennung.html#no_permission_userprofile (besucht am 18. 09. 2020).

[111] WIPO. *Global Brand Database*. URL: https://www3.wipo.int/branddb/en/ (besucht am 06. 01. 2021).

[112] wxPython. *wx.media.MediaCtrl*. URL: https://wxpython.org/Phoenix/docs/html/wx.media.MediaCtrl.html#wx-media-mediactrl (besucht am 22. 10. 2020).

[113] zszyellow. *GitHub - WER-in-python*. URL: https://github.com/zszyellow/WER-in-python (besucht am 25. 11. 2020).

[114] Zulko. *MoviePy*. URL: https://zulko.github.io/moviepy/ (besucht am 15. 11. 2020).

A Anhang

A.1 Brainstorming

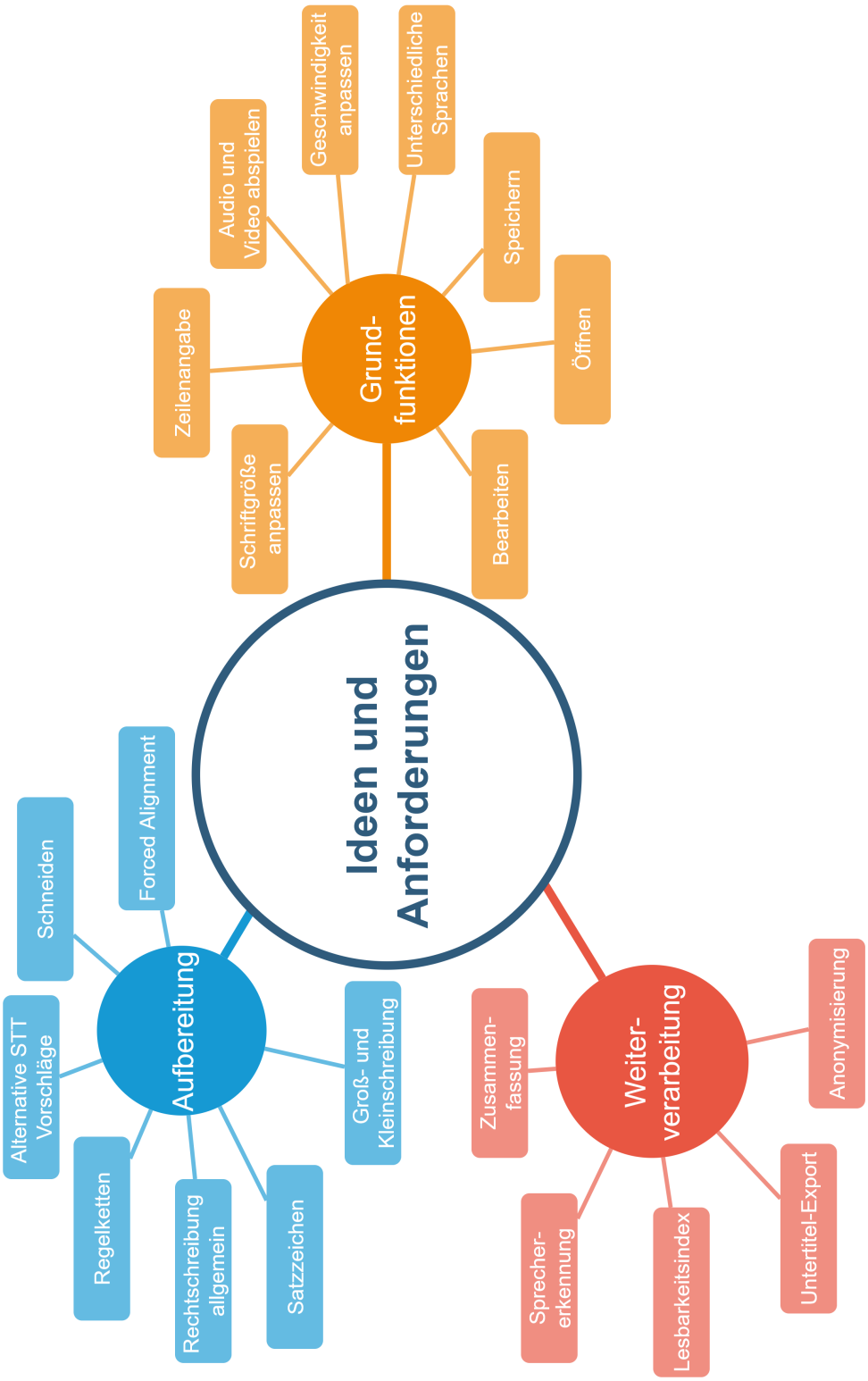


Abbildung 60: Brainstorming: Ideen und Anforderungen

A.2 Komponentendiagramm

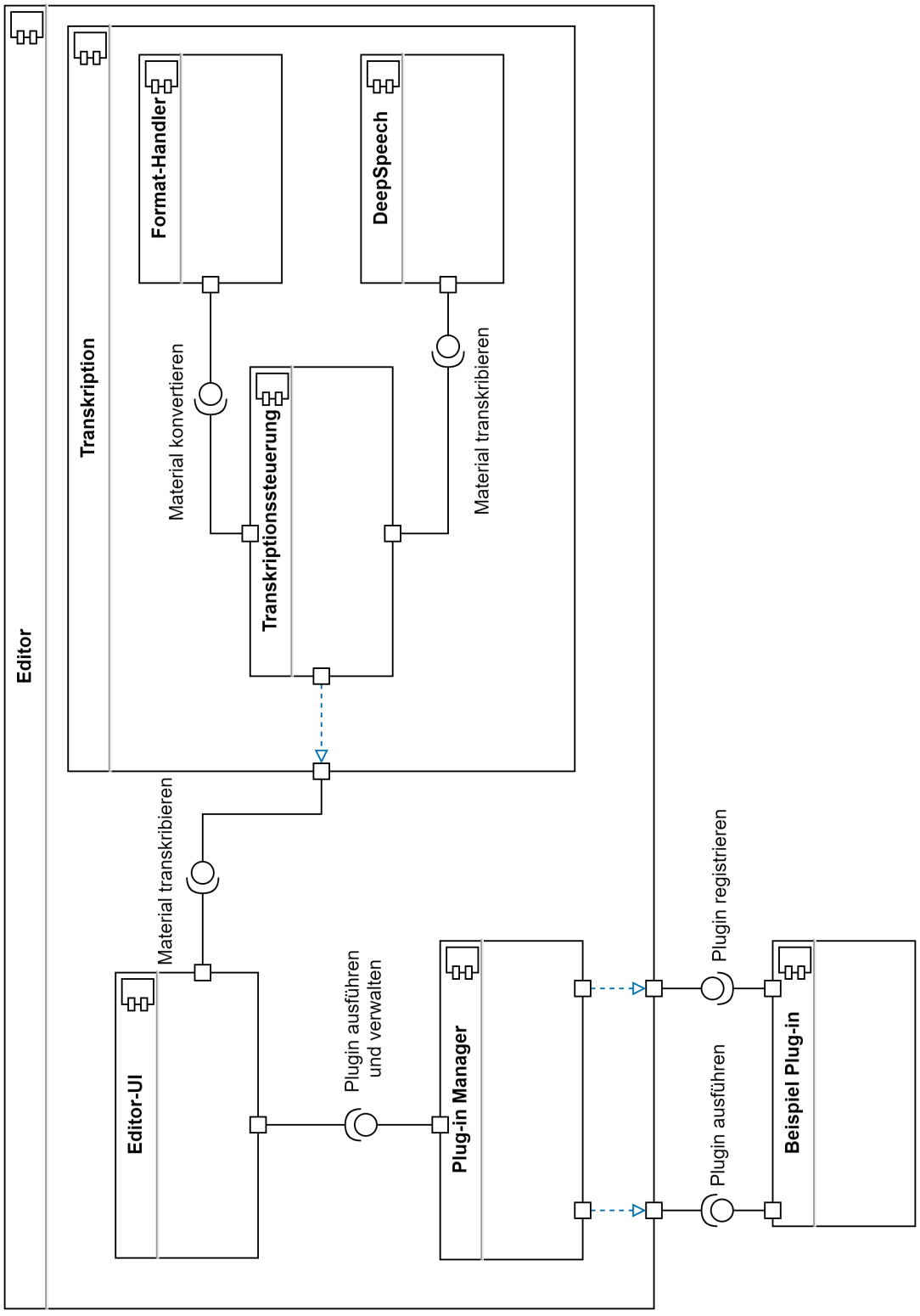


Abbildung 61: Komponentendiagramm des zu entwickelnden Editors

A.3 Editor-Maske

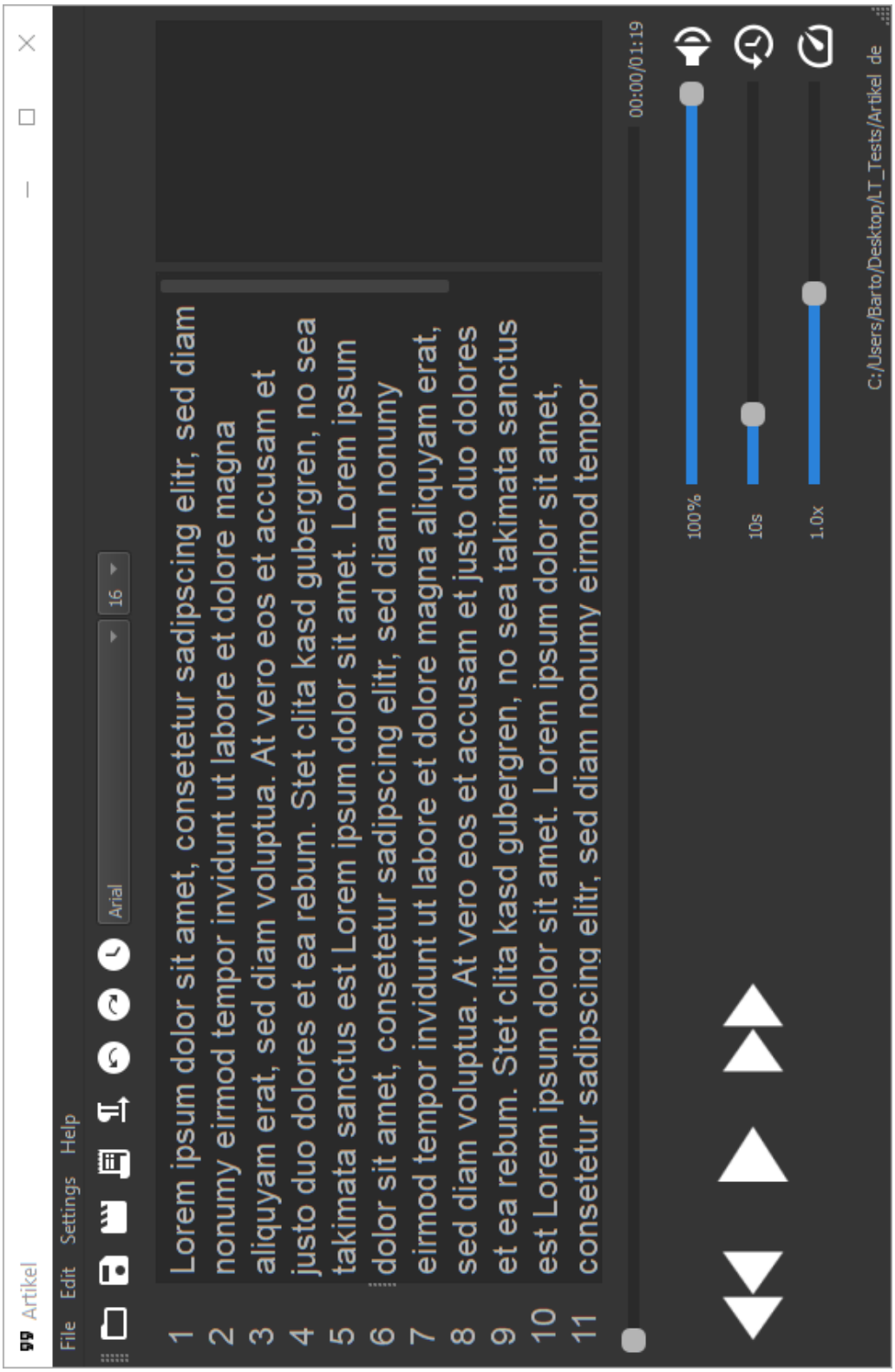


Abbildung 62: Editor-Maske

A.4 Fehlerarten des Beispieltextes

Original	DeepSpeech	Art
als		D
hand		D
schmieren	handspiel	S
smeergeld	mehr	S
	geld	I
den	dem	S
den	dinge	S
gewinn	wenn	S
vier		D
abs		D
fünf		D
nr	absender	S
zehn	meister	S
einkommensteuer-gesetz	gesetz	S
dass	das	S
aufwendungen	aufwendung	S
ahndung	ahnung	S
geldbuße	geldbusse	S
fuer		D
zwecke	der	S
des	zweckmaessigen	S
besteuerungs-verfahrens	verfahren	S
steuerstraftaten	steuergeraeten	S

Original	DeepSpeech	Art
steuerordnungs-widrigkeiten	steuer	S
	ordnungs-widrigkeiten	I
	auszug	I
	ende	I
paragraphen	paragraph	S
stgb		D
bei		D
fahrgast		D
zahlen	fahrgastzahlen	S
regelmaeßig	regelmaessig	S
fraßen	fassen	S
postillon	posten	S
schnellerem	schnelleren	S
denn		D
er		D
musste		D
seinen		D
fahrplan	der	S
einhalten	emotionalen	S
postillone	alten	S
und	bastionen	S
einzelner	einzelne	S
waeren	waere	S

Tabelle 9: Fehlerarten der Transkription des gesprochenen Schmiergeld-Artikels. [109], [57]

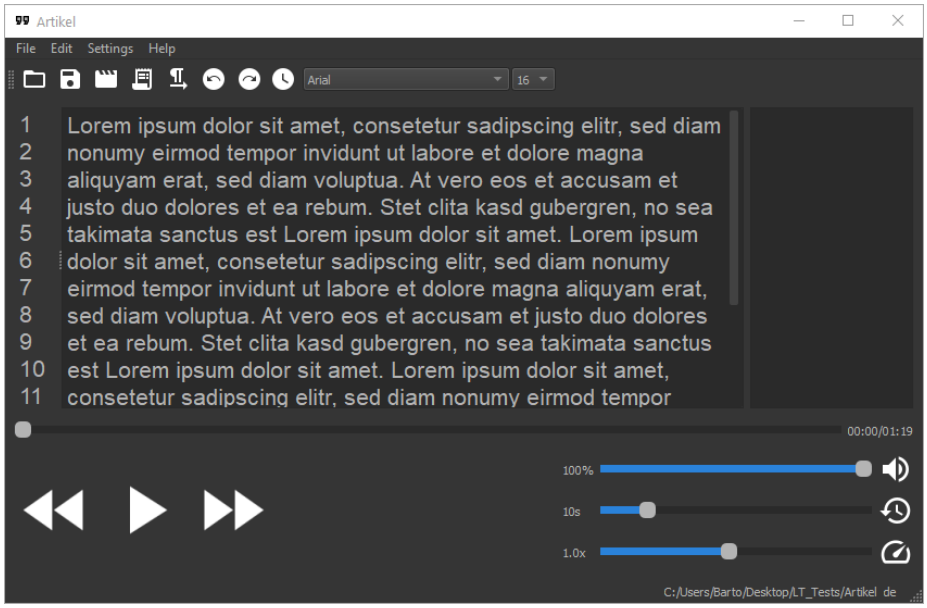
A.5 Einführung / Anleitung

Help / Manual

L A Z Y T R A N S C R I P T

Open source editor

for transcribing audio and video material



Contents

List of figures **2**

1 Introduction **3**

2 Prerequisites **3**

3 Start Dialog **3**

4 Create a Project **4**

5 Open a Project **4**

6 Editor GUI **5**

6.1 Menubar 5
6.2 Toolbar 6
6.3 Textarea 6
6.4 Media-Controls 7
6.5 Word by word editing mode 7
6.6 Settings 8
6.7 Text modules 9

7 (Un)installing a plug-in **9**

References **10**

List of figures

1 Start Dialog 3
2 Creating a new Project 4
3 Editor GUI 5
4 Menubar 5
5 Toolbar 6
6 Textarea 6
7 Media-Controls 7
8 Word by word editing mode 8
9 Settings 8
10 Text modules 9

1 Introduction

This manual aims to introduce you to the basic functionality of LazyTranscript. LazyTranscript is an open source editor for transcribing audio and video material. It was developed in the context of a master thesis and is based on the open source speech-to-text engine "DeepSpeech" [3] from Mozilla.

2 Prerequisites

To use LazyTranscript, you must first download a DeepSpeech model of your language. A good source for this is: https://discourse.mozilla.org/t/links-to-pretrained-models/62688. Then you have to paste the downloaded `.scorer` and `.pbmm` files into

`<LAZYTRANSCRIPT_FOLDER>/models/<LANGUAGE_ABBREVIATION>`.

The abbreviation usually consists of the language and the region. For example:

1. de-DE: German in Germany
2. en-US: English in the United States
3. fr: French

For more information see: [1], [2]

3 Start Dialog

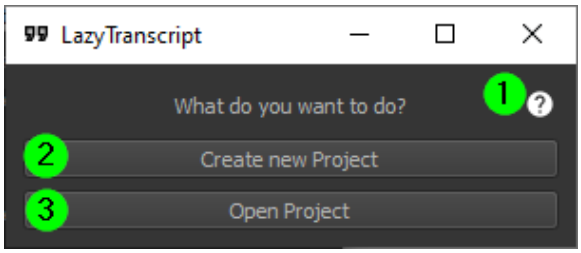


Figure 1: Start Dialog

This window opens when you start the program. The numbered elements perform the following actions:

1. Opens this manual in the system's default PDF viewer.
2. Create a new LazyTranscript project. (Section 4)

3. Open an existing LazyTranscript project. (Section 5)

4 Create a Project

When selecting the "Create new Project" option, the following window is shown.

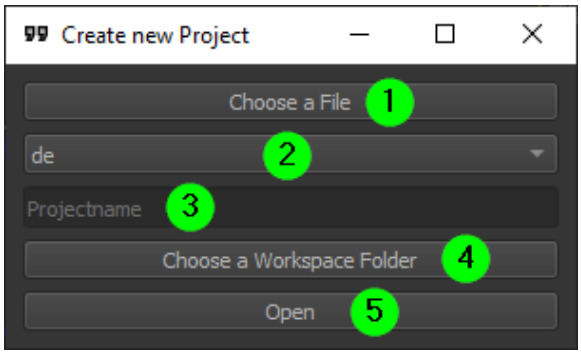


Figure 2: Creating a new Project

The numbered elements perform the following actions:

1. A file dialog opens and the audio or video file that should be transcribed must be selected.
2. Here the language of the file must be selected. A drop-down menu automatically displays the available languages whose model directories were created in Section 2.
3. Choose a project name
4. A dialog opens and the workspace folder must be selected. This folder can contain different projects. The program will create a directory with the previously selected project name, in which the project-relevant files (for example the transcription itself) are stored.
5. Starts the creation process. This includes createing the project folder and the initial transcription. A progress bar shows the current progress. When the process is finished, the editor GUI opens automatically (chapter 6)

5 Open a Project

If the “Open Project” option is chosen, a dialog appears in which the project folder must be selected. The editor GUI (Section 6) will be opened afterwards.

6 Editor GUI

When you open a project, the window shown in Figure 3 appears. In the following chapters, the different areas of this window are explained in more detail.

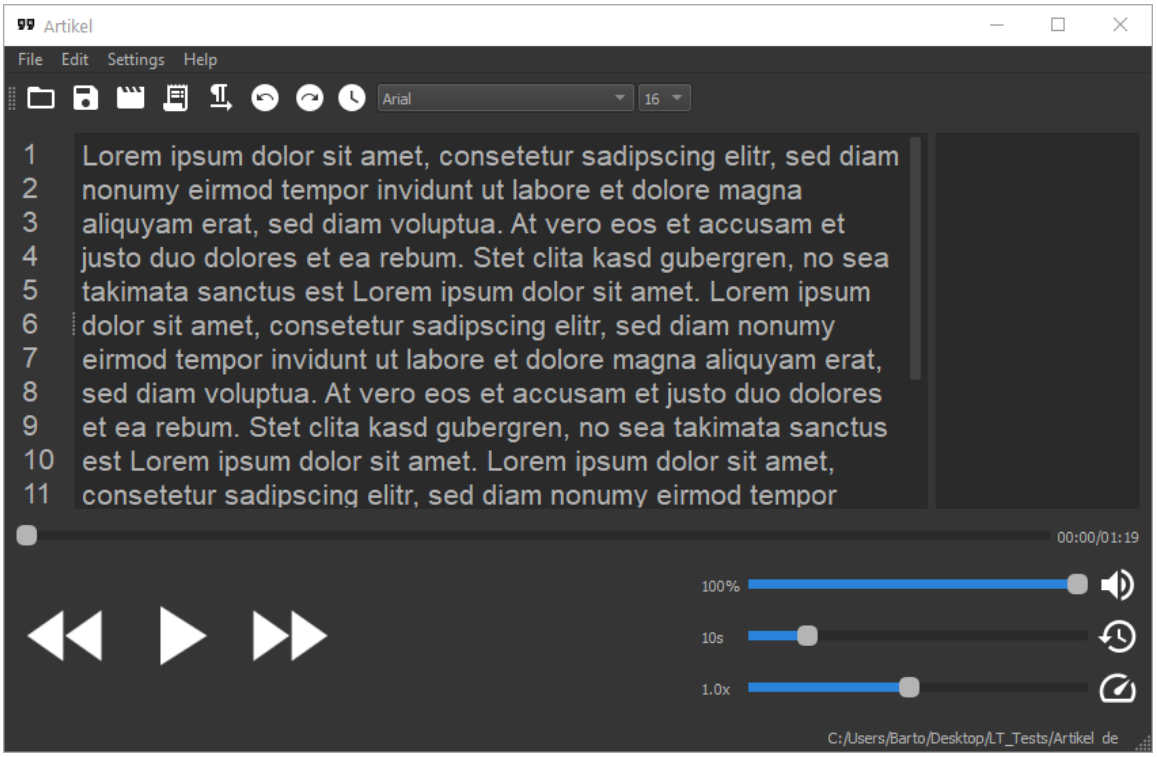


Figure 3: Editor GUI

6.1 Menubar

Figure 4: Menubar

The numbered elements in the menubar contain the following functions:

1. File: New, Open, Save [CTRL+S]
2. Edit: Undo, Redo, Copy, Paste
3. Settings: Open Settings (Section 6.6), Text modules (Section 6.7)
4. Help: Open Help [F1] (this Manual), Licences

6.2 Toolbar

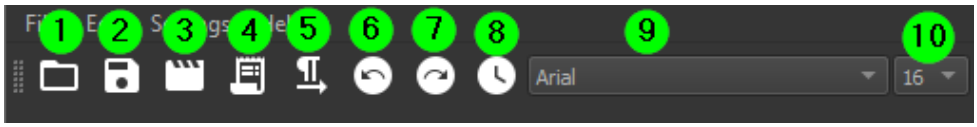


Figure 5: Toolbar

The numbered elements perform the following actions:

1. Open Project
2. Save Project [CTRL+S]
3. Open Videooutput (works only if the source material was a video)
4. Activate Text modules (Section 6.7)
5. Show special characters like ¶ for linebreak or · for whitespace
6. Previous word in the word by word editing mode [F5] (Section 6.5)
7. Next word in the word by word editing mode [F6] (Section 6.5)
8. Inserts the current timestamp of the media player using the following format: [hh:mm:ss]
9. Change font
10. Change fontsize

6.3 Textarea

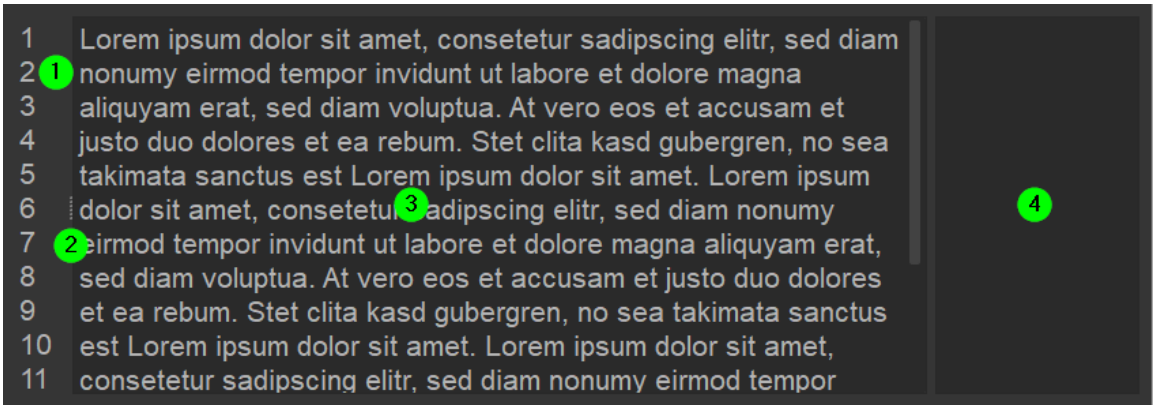


Figure 6: Textarea

The numbered elements contain the following information or provide the following functionality:

1. Line numbers
2. Anchor to increase or decrease the size of line numbering
3. Text field itself, contains the transcription
4. List of Buttons in the word by word editing mode (Section 6.5)

6.4 Media-Controls

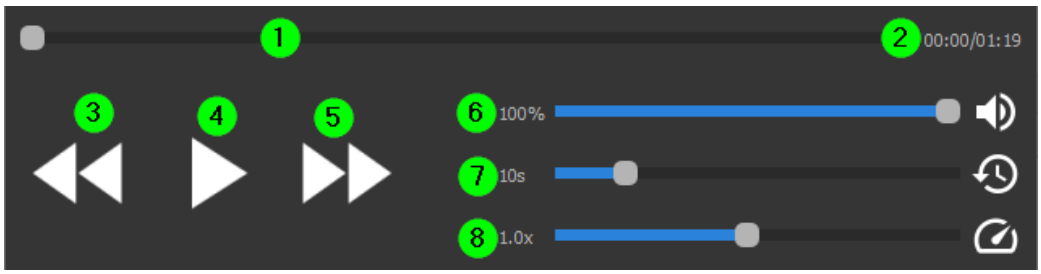


Figure 7: Media-Controls

The numbered elements allow you to perform the following actions or change the following:

1. Seek bar which displays the current elapsed time / total duration
2. Played / total time
3. Rewind by x seconds [F2]
4. Play / Pause [F3]
5. Fast forward by x seconds [F4]
6. Volume control
7. Rewind and fast forward duration
8. Playback speed

6.5 Word by word editing mode

The word by word editing mode is activated with the F6 key or the respective toolbar button. As can be seen in Figure 8, the individual words are highlighted (1). The F5 or F6 key can then be used to move to the previous or next word. In the list next to the text, there are buttons for

correction or further processing. Except for the option to hear individual words in the source material, the buttons are provided by plug-ins.

The workflow for editing could be the following: F6 -> Execute correction -> F6 -> Execute correction -> ...

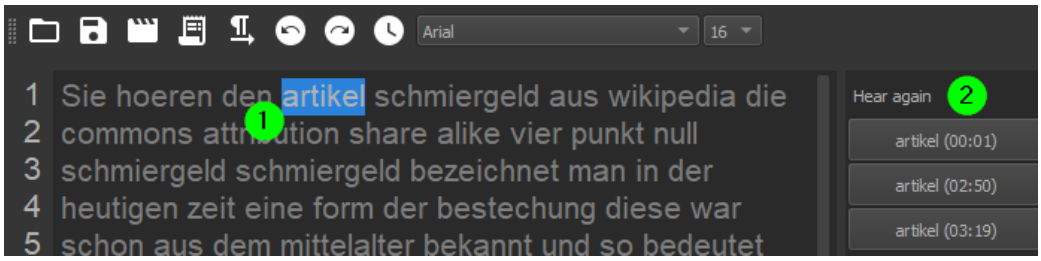


Figure 8: Word by word editing mode

6.6 Settings

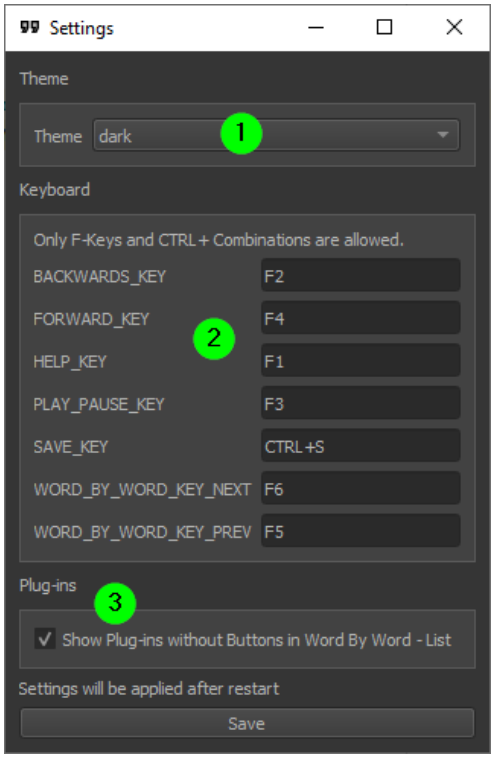


Figure 9: Settings

The numbered areas let you change the following options:

1. Choose between a dark and a light Theme

2. Customization of shortcuts and key combinations
3. Because it is possible that plug-ins do not define buttons, displaying the plug-in's message and headline can be disabled for plug-ins that do not define buttons.

6.7 Text modules

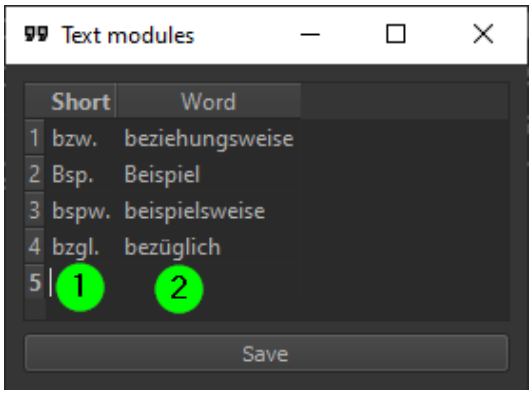


Figure 10: Text modules

Here the text modules can be defined. For this purpose, only an abbreviation (1) and the word (2) by which it should be replaced must be defined. In order to make the replacement work, it must be activated via the toolbar button as mentioned in section 6.2.

7 (Un)installing a plug-in

In order to install a plug-in, its directory must simply be copied into the plugins directory (`<LAZYTRANSCRIPT_FOLDER>/plugins/`). Vice versa, a plug-in can be uninstalled if the particular directory is deleted again.

References

[1] M. Davis A. Phillips. *RFC5646 - Tags for Identifying Languages*. 2009. URL: https://tools.ietf.org/html/rfc5646 (visited on 12/28/2020).

[2] Richard Ishida. *Language tags in HTML and XML*. W3C. 2014. URL: https://www.w3.org/International/articles/language-tags/ (visited on 12/28/2020).

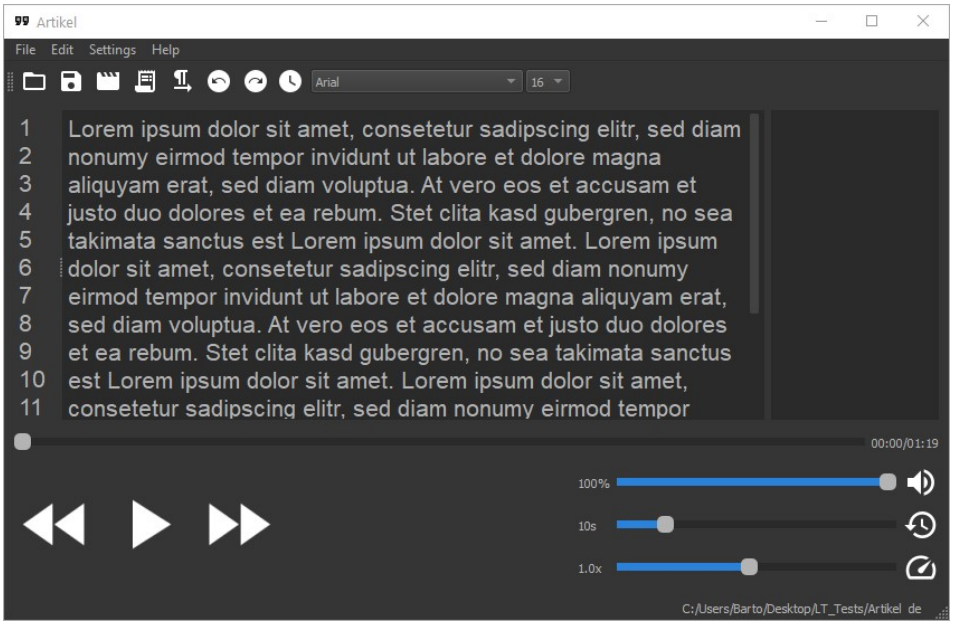
[3] Mozilla. *GitHub - Project DeepSpeech*. URL: https://github.com/mozilla/DeepSpeech (visited on 09/23/2020).

A.6 Readme

LazyTranscript

Description

LazyTranscript was developed in a master thesis and tries to simplify the transcription of audio and video material with the help of DeepSpeech and various plug-ins. Through various plug-ins, the editor tries to reduce the effort required to correct the errors in the transcription generated by DeepSpeech.



Installation and usage

0. Read the manual
1. Install Python 3.8
2. Install the requirements

```
pip install -r requirements.txt
```

3. Remove unwanted plug-ins from the plug-ins directory. Look at the next section for more details about the plug-ins.
4. For each remaining plug-in, run the following command in the respective plug-in folder

```
pip install -r requirements.txt
```

5. Install a DeepSpeech model, see: section 2 in the manual
6. Start LazyTranscript with

```
python main.py
```

7. Edit the Transcription manually or use the word by word editing mode, see manual section 6.5

Preinstalled plug-ins

1. word: This plug-in allows you to prepend, append, capitalize, replace, concat or remove words in the word by word editing mode.
2. words_to_number: This plug-in converts words to the number which they represents. Supports only english and german.
3. add_special_character: This plug-in allows you to append {. , ! ? ( )} to each word in the word by word editing mode.
4. lemmatization: This plug-in adds a toolbar-button which try to capitalize every noun. Currently supports only german.
5. deepspeech_alternatives: This plug-in shows alternative words from deepspeech in the word by word editing mode.
6. vosk_alternatives: This plug-in shows alternative words from vosk in the word by word editing mode. In order to use this plug-in you need to download a model for your language and unzip it in the vosk_alternatives/model folder.
7. umlauts: This plug-in adds a toolbar-button which try to replace all spelled out umlauts
8. language_tool_correction: This plug-in shows correction from a locally running languagetool-server. In order to use this plug-in you need to download the language-server and unzip it in language_tool_correction/language_tool/server. To improve the results, the n-gram models can optionally be installed. They simply have to be unpacked to language_tool/n-gram/<LANGUAGE_TAG>.
9. summary: This plug-in adds a toolbar-button that displays a summary of the current transcription.
10. fake_data: This plug-in shows fake data in the word by word editing mode. This could be useful to remove any personal information.
11. readability: This plug-in adds a toolbar-button that display the current readybility after flesch-reading-ease.

Support

If you find any problems or need help, just open an issue.

Roadmap

In the future i would like to implement

1. Forced alignment, to map the transcription to the audio
2. SRT export to create subtitles from the transcription

Contributing

If you want to make some improvements or create a own plug-in: see CONTRIBUTING.md for more information.

Acknowledgment

Thanks to DANBER for the awesome deepspeech models.

License

A.7 Contributing

Contributing

Introduction

Thank you for liking LazyTranscript so much that you are considering contributing to it. It's only through people like you that LazyTranscript can continue to evolve and make transcriptions as easy as possible!

Through these guidelines, all helpers of this project should benefit from each other's contribution. The guidelines try to solve common questions and problems in advance and try to reduce the workload for everyone.

LazyTranscript is happy about every contribution! You can for example:

- improve or extend the documentation
- write your own plug-in (little explanation follows later in this document)
- make code improvements
- add features
- help others

Feel free to suggest or implement ideas!

Ground Rules

In order to facilitate the collaborative work on LazyTranscript, the following basic rules should be followed.

Follow the CODE_OF_CONDUCT!

- New functionality should preferably be implemented as a plug-in.
- Fill issues not only with error messages, but also with examples.
- The more detailed and extensive the issues are, the easier they are to track.
- Create a new branch for each new feature.
- Features should be as isolated as possible.
- If you are unsure about an idea, feel free to create an issue to discuss your idea and find contributors.
- External programs, which have to be installed without pip, should be avoided urgently. This is to prevent that the installation barrier of LazyTranscript becomes too large.
- Issues should not be used for basic python questions.

Your First Contribution

If you want to help with LazyTranscript, but are not sure what to do, it is worth taking a look at the issues. If possible, these are tagged so you can quickly determine that they match your expertise.

Working on your first Pull Request? You can learn how from this *free* series How to Contribute to an Open Source Project on GitHub

Once the first issues are established, it is planned to create and support "first-timers-only" issues according to http://www.firsttimersonly.com/

Now you can start creating your first changes. Feel free to ask for help!

Getting started

1. Create your own fork of the code
2. Do the changes in your fork
3. Create a Pull Request

If you find a small fix and don't want to correct it yourself, create a beginner friendly issue so people new to open source can fix them.

Code, commit message and labeling conventions

- Class names should use the CapWords convention
- Function and variable names should be lowercase separated by underscores
- You can find a code documenation in the "docs" folder.

Code review process

The maintainer(s) try to process and comment pull requests as fast as possible. However, do not forget that LazyTranscript is a hobby project and therefore delays are possible.

Quick guide for creating your own plug-in

1. Create a folder under the "plugins" directory for your plug-in
2. In this folder create a python-file called "<YOUR_PLUGIN_NAME>_lt_plugin.py"
3. Create a class called "plugin" which inherited the "IPlugin"-Class
4. Override the necessary methods from IPlugin and use the plugin_manager to manipulate or receive the text or specific words
    - If you want to make a toolbar-plug-in then you need override the "get_toolbar_action"-Method and create the QActions there
    - If you want to make a word by word editing plug-in then you need to override the "get_word_action"-Method and return the QPushButtons via the "add_new_word_by_word_action"-Method of the plug-in manager.
5. Override the get_name Method and return your plug-in name

Optional:

- Override the project_loaded-Method to do something when the project is loaded
- For long task you should use a QThread and use the signals in the IPlugin-Class to notify the main thread
- If you are using some open source libraries you should include them in the "licence"-directory in your plug-in directory.

If you have further questions: Have a look at the other plug-ins, read the documentation or feel free to create a issue with your question.

How to report a bug

If you find a security vulnerability please send it to: maurice(at)samuel-bolle.de

If not: Create an Issue and try to describe the bug as detailed as possible. As the project evolves and certain patterns are found in the bug reports, an ISSUE_TEMPLATE will be generated.

How to suggest a feature or enhancement

Create an issue and try to describe the feature and its gains as detailed as possible. If you already have an idea for a possible implementation, describe it as well. This way, others can evaluate this idea and refine the implementation.

Erklärung über die selbständige Abfassung der Arbeit

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.
Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

(Ort, Datum, Unterschrift)